

(19)



Europäisches Patentamt
European Patent Office
Office européen des brevets



(11)

EP 0 998 076 A1

(12)

EUROPEAN PATENT APPLICATION

(43) Date of publication:

03.05.2000 Bulletin 2000/18

(51) Int Cl. 7: H04L 12/26, H04L 12/24

(21) Application number: 99308205.6

(22) Date of filing: 18.10.1999

(84) Designated Contracting States:

AT BE CH CY DE DK ES FI FR GB GR IE IT LI LU

MC NL PT SE

Designated Extension States:

AL LT LV MK RO SI

(30) Priority: 30.10.1998 US 184101

(71) Applicant: LUCENT TECHNOLOGIES INC.

Murray Hill, New Jersey 07974-0636 (US)

(72) Inventors:

- Foley, Michael P.
Elmwood Park, Illinois 60707 (US)
- Vangsness, Kurt A.
Aurora, Illinois 60506 (US)

(74) Representative:

Buckley, Christopher Simon Thirsk et al

Lucent Technologies (UK) Ltd,

5 Mornington Road

Woodford Green, Essex IG8 0TU (GB)

(54) Method for controlling a network element from a remote workstation

(57) A method is provided for controlling a network element from a client at a remote work station connectable to the network, the network, element is registered

for attributes to be tracked, and attributes associated with the network element are polled only if the client requests the monitoring of the network element.

BEST AVAILABLE COPY

EP 0 998 076 A1

Description**Background Of The Invention**

5 [0001] This invention relates, generally, to a method for controlling a network element and, more particularly, to a method for remotely controlling the network by communications through the network.

[0002] Network management systems in which network elements, or management agents, are remotely controlled from a remote, management work station by means of communications between the management work station and the managed network elements sent through the network are known. Such known network management systems 10 employ a special communication protocol for communications between the remote work station running a management program and an element management server that contains a management information base that defines the interface between the work station and the network elements.

[0003] Known systems such as (Hewlett Packard HP-OV NNM or DM, Sun Microsystems Solstice) present an interface where the client application must poll the network element when status is needed. In these systems, the polling 15 may not be coordinated and is replicated for each client, if each client is interested in the same attributes. Also, each of the clients receive the full results for each polling cycle (even if there was no change from the last cycle), increasing the bandwidth used to communicate between the client application and the network element, as well as creating additional processing overhead due to the replicated polling at the network element.

Summary Of The Invention

[0004] A method is provided for controlling a network element from a remote work station connectable to the network. The method provides for registering the network element for attributes to be tracked, and polling for attributes associated 25 with the network element only if the client requests the monitoring of the network element. Changes in attributes are reported when the client requests notification of changes in attributes. For attributes polled for a plurality of clients, changes in the attributes to one of the plurality of clients requesting notification of changes in the attributes are reported.

[0005] The method further provides for polling once for a plurality of clients that registers for the same attributes and reporting asynchronously changes in the attributes to a plurality of clients.

[0006] Another aspect of the invention provides for running an object oriented program at the remote work station 30 to control an object associated with the controllable network element, translating interface operations generated by the work station during the running of the object oriented program to corresponding translated interface operations in an object oriented language associated with the object being controlled, and connecting the corresponding translated interface operations through the network to an object server to control the object associated with the network element in accordance with the translated interface operations.

[0007] These and other features and advantages of the present invention will become apparent from the following 35 detailed description, the accompanying drawings and the appended claims.

Brief Description Of The Drawings

40 [0008] The foregoing advantageous features will be described in detail and other advantageous features of the invention will be made apparent from the detailed description of the preferred embodiment of the invention that is given with reference to the several figures of the drawings, in which:

45 Fig. 1 is a functional block diagram of the preferred embodiment of a management system using the preferred network element control method of the present invention;

Fig 2 is a functional block diagram of the preferred embodiment of the translating interface shown as a single functional block in Fig. 1;

50 Fig. 3 is a functional block diagram illustrating the interface with the controlled network element that is visible to object oriented client management application at the work station of Fig.1;

Fig. 4 is a table of a plurality of service objects that interact with the client management application run at the work station of Fig.1;

55 Fig. 5 is a table of a plurality of call back functions performed at the translating interface of Fig.2;

Fig. 6 is a table of the different fundamental data types capable of being translated by the translating interface of

Fig.2 in accordance with the invention.

Fig. 7 is a block diagram showing the relationship between client application-specific service object, and the internal service representative of managed object instances;

Fig. 8 is a table summarizing filter criteria that is valid for each event category; and

Fig. 9 is a table defining specific exceptions with an EMAPI exception code containing one of the listed values.

10 Detailed Description

[0009] This invention provides an application programming interface (API) and protocol that provides for efficient communication between a distributed client application and an element management server independent of the communication protocol to the network element. The Element Management Application Programming Interface (EMAPI) provides the following benefits over known management system. The invention has application in the management of a telecommunication network element. For more information regarding such a management of a telecommunication network element refer to commonly owned U. S. patent application serial number 09/088,463, entitled "Method for Computer Internet Remote Management of a Telecommunication Network Element" by William E. Barker, Lisa M. Connelly, Marvin A. Eggert, Michael P. Foley, Kenneth R. Macfarlane, Philip M. Parsons, Girish Rai, Jerome E. Rog, and Kurt A. Vangness, filed on May 31, 1998, the disclosure of which is hereby incorporated by reference.

- Efficient use over low bandwidth connections. Client applications register for network element information they wish to track and after an initial set of data only receive incremental updates (deltas) when there are changes.
- Centralized polling of attributes. Attributes are only polled if a client exists that has registered to monitor the attribute. If multiple clients register for the same attribute(s), the polling is not repeated for the clients-only a single polling cycle is performed.

[0010] The invention is used in an operations, administration and maintenance system 20 as shown in Fig. 1. The system 20 includes a PC or workstation 22, an element management server (EMS) 24, an interface in accordance with the invention 26, located between the workstation 22 and an object server 25. An application processor 28 is connected to the element management server 24.

[0011] The workstation 22 includes a web browser 30 which is the interface to the client and is a host for JAVA applets 32 and web browser HTML 35 which is a hypertext markup language.

[0012] The system 20 operates on a cluster computing environment, and leverages off-the-shelf technology to enable additional customers visible features, while extending to subsequent releases and other projects, with minimal increased cost. System 20 is provided through the web browser interface and a SNMP based element management platform.

[0013] A client executes applications via web pages at the workstation 22. The client makes requests for various views of the network status by making selections through the web browser 30. The web browser requests pages from the web server 28 which transmits HTML pages that contain instructions to load and run the appropriate JAVA applet 32. Once the applet starts, it communicates with the object server 25 through the interface 26 to perform initialization and to request initial configuration and status information that is appropriate for the current requested view. The JAVA applet 32 then registers with the object server 25 for subsequent notifications of changes to configuration and status that it requires to keep the view up to date. The client may perform commands to request various maintenance operations on the network element 28. These commands are converted into appropriate requests through the interface 26 and perform operations on the object server 25. The commands are then translated into SNMP and are transmitted to the network element 28 through the SNMP library 33. Acknowledgements and command responses from the network element 28 are transmitted through the SNMP library 33, are converted to events by the object server 25 and transmitted to originating JAVA applet 32 through the use of callbacks defined by the interface 26.

[0014] In one embodiment of the invention, as shown in Fig. 3, client applications in JAVA applets 32 include an active alarm list browser, a system alarm survey and a network element detailed status display. Client applications communicate with the web server 28 via the interface 26 in accordance with the invention, to the element manager through a distributed object request architecture such as CORBA. The interface 26 provides a constant interface to all managed objects in the network, and hides the implementation details associated with the element manager platform.

[0015] The interface 26 (EMAPI) is the definition of objects, attributes and operations that comprise the protocol used between client applications and the server to manage network elements. The EMAPI uses the industry standard CORBA to provide distribution of the objects and their operations and to allow for the implementation of the client and server

to be in different programming languages and on different computer architectures.

[0016] The client interface to the server and the managed object attributes is described in the interface 26 and managed object notation provides a consistent model of all managed objects in the network, hiding the implementation details associated with the element manager platform from client applications, thus clients do not need to know the underlying protocol to the network elements. Managed objects specific logic is encapsulated within the managed object instead of scattered throughout various applications thus simplifying client application development.

[0017] Each physical, selected non-physical and logical component in the network is modeled as a managed object, which the Server makes visible to distributed client applications through the facilities of the Common Object Request Broker Architecture (CORBA). EM clients need only be concerned about the attributes and operations defined for each application managed object, and not the details of network-level protocol and the server infrastructure required to support object services.

EMAPI Object Definition

[0018] Fig. 3 illustrates all of the interfaces visible to client applications which does not depict process or processor boundaries, which are made transparent by the client and server object request brokers (ORBs). Application services are provided through object interfaces formally defined in the CORBA Interface Definition Language (IDL). The IDL specification of the interfaces described in this document is provided in the Appendix A.

[0019] The service objects resident on the server with which client applications will interact are shown in Fig. 4.

[0020] Client applications which register for real-time status updates or notification of events, alarms or configuration changes must provide a reference to a local callback object which the server will use to propagate information asynchronously. The callback interfaces defined in the interface 26 are shown in Fig. 5. Classes which implement these interfaces must be defined and instantiated in client code.

Data Representation

[0021] There are several fundamental data types defined in the interface 26, which fall into one of the two categories shown in Fig. 6.

Session Management

[0022] Each EM client session is logically associated with a unique login-host combination. Multiple client applications may be associated with the same session, though only one need be registered for the session to be considered active. Session and application identifiers are assigned by the User Session Manager to track resources used by the client, and in future releases, to correlate client access permissions with operation requests. Applications may or may not cross process boundaries. For example, multiple instances of the EMS Command Line Interface (CLI) application registered with the same login and host name will share the same session id, but each process is associated with a different application id. In the EMS Graphical User Interface, all application frames execute in the same process space (albeit in different Threads), yet each frame is associated with a distinct application id. Note that each client application is required to independently register a periodic heartbeat to validate for the Server that its associated resources are still needed.

[0023] The UserSession service object provides the following interfaces:

- **startApplication**
This method must be invoked for each client application initialization.
- **stopApplication**
A client invokes this method to notify the Server that a target application is terminating, and its associated resources should be released.
- **stop**
This method may be used to deregister all applications associated with the same session identifier.
- **heartbeat**

[0024] This method must be invoked at least every UserSession::HeartbeatPeriod seconds to avoid a timeout condition which, when detected by a Server audit, will result in the release of all resources utilized by an application.

[0025] Refer to the description of interface UserSession in the attachment for additional details.

Managed Objects

[0026] A managed object (MO) is an abstract representation of a physical or logical resource which may be managed by the EMS, such as a network element, maintenance unit or data link. The EM Server will implement one application-specific service object for each type of physical or logical resource to be managed. Each of these service objects defines a set of attributes which identify managed object properties, as well as the operations which may be performed on a specified managed object instance. (The decision to provide access to instance information through a single "service object" stems from the fact that current ORB implementations become unstable when managing very large numbers of remote references.) Fig. 7 depicts the relationship between Client, application-specific service object, and the internal Server representation of managed object instances.

[0027] Each managed object service class is uniquely identified by a ClassCode. Each managed object instance is uniquely identified by an InstId. Any object instance in the system may be uniquely referenced by a managed object identifier (Oid), which is the combination of ClassCode and InstId.

[0028] Managed object status information is reported by a service object as a sequence of attribute code-value pairs.

[0029] Configuration information is reported as a sequence of ConfigData structures, which are defined to contain:

- network element instance id
- managed object instance id
- a managed object key list reported as a sequence of attribute-value pairs-- when length is greater than 0, the key list specifies the associated logical identifiers (LogicalIds)

[0030] Each managed object service class must implement the MO interface, which defines the following configuration and status services:

- **viewConfig**
A client uses this method to obtain the current EMS view of the managed object configuration for a specified network element instance. Note that the reserved instance identifier AnyInstance may be used to obtain configuration information for all network elements.
- **notifyConfig**
A client may also register for an initial view of managed object configuration information and notification of subsequent changes via callback. The initial view is returned with a notification type CONFIG_INIT. Subsequent changes are reported with type CONFIG_CREATE or CONFIG_DELETE.
- **cancelNotify**
A client uses this method to cancel registration for managed object configuration notifications associated with a specified client application.
- **getPersistent**
A client may use this method to retrieve the set of attribute codes (SeqAttrCode) identifying all "persistent" data maintained by this service object. Values for persistent attributes of each managed object instance are stored and kept current irrespective of any client requests.
- **getAttrSpec**
A client may use this method to retrieve the name and codes of all attributes defined for a target service class (currently used for debugging only).
- **getKeySpec**
A client may use this method to retrieve the set of codes (SeqAttrCode) identifying the attribute(s) which represents the logical identifier(s) of any instance of the target class.
- **viewStatus**
A client may invoke this method to obtain the EMS view of the current values for a specified set of persistent attributes for a specified managed object instance.

- **getStatus**
A client may use this method to register for a snapshot of current status information. This interface differs from the previous one in that the requested attribute list may specify any managed object attribute codes--not just those associated with persistent data, and the information is returned via client status callback (StatusCB).
- **startUpdate**
A client may also register for an initial view and notification of any updates to a list of selected attributes for a specified managed object instance. In this case, an initial view is reported via client callback with a notification type STATUS_INIT. Subsequent changes are reported with type STATUS_CHANGE. Note that managed object instance deletions are reported only through configuration change notification to avoid a potential flurry of client status callbacks when a network element is unequipped.
- **stopUpdate**
A client uses this method to cancel registration for managed object status updates associated with a specified client application.
- **getInst**
A client may use this method to obtain a managed object instance identifier for a specified network element instance id and managed object key list.

[0031] Note that each method requires a client session application identifier (SessionAppId) to validate user access. In the case of configuration or status change notification registration, this identifier is also used to keep track of the additional server resources utilized while the client application is active.

[0032] Refer to the description of interfaces MO, ConfigCB & StatusCB in the attachment for additional details.

Network Element Level Managed Objects

[0033] Each network-element level managed object must also implement the NEMO interface which defines additional network-element level configuration services:

- **viewNEconfig**
- **[0034]** A client may invoke this method to obtain the current EMS view of the network element configuration.
- **notifyNEconfig**
A client may also register for an initial view of network element-level managed object configuration information and notification of subsequent changes via callback. The initial view is returned with a notification type CONFIG_INIT. Subsequent changes are reported with type CONFIG_CREATE or CONFIG_DELETE.
- **cancelNEnotify**
A client application should use this method to cancel registration for network element managed object configuration updates.
- **getNEinst**
A client may invoke this method to retrieve the NEMO instance identifier of the network element associated with a specified logical id.
- **getLogicalId**
A client may invoke this method to retrieve the logical identifier of the network element associated with a specified NEMO instance id.
- **getContainment**
A client may invoke this method to obtain a sequence of containment information for the target NEMO, where each entry in the sequence contains the name, class code and CORBA reference to a contained service class object.

[0035] Note that each method requires a client session application identifier to validate user access. In the case of configuration change notification registration, this identifier is also used to keep track of the additional server resources

utilized while the client application is active.

[0036] Refer to the description of interfaces NEMO & NEconfigCB in the attachment for additional details.

Descriptive Entity Objects

[0037] Application objects of this type are defined to provide type and attribute information for abstract entities, such as data communicated between the EMS and network elements which are not part of a managed object description (e.g. SNMP trap definitions and command groups). Descriptive entity objects provide no implementation-they are defined in application-specific IDL and known by client applications at compile time.

Event Distributor

[0038] An event is reported as a combination of the following:

1. A header, which contains information of most general interest:

- Time of the event
- Event category defined to be one of the following:

• Alarm Set

• Alarm Clear

• Command Acknowledgment

• Command Response

• Configuration Change

• Informational Message

• Initialization

• State Change

- Network element object identifier

• Network element alarm level-meaningful only for alarm set

• Maintenance unit object identifier (if applicable)

• Maintenance unit alarm level-meaningful only for alarm set

• A command identifier (CmdId) defined as a user session id & command sequence number-meaningful only for command acknowledgment & response.

1. Event data defined as a sequence of structures which contain:

• A ClassCode of a managed object, network element or descriptive entity

• A sequence of attribute code-value pairs

[0039] Client applications may request a copy of the event stream, as processed by the event distributor, filtered on information specified in the event header. Filter wildcards are implemented with "out-of-band" values:

- Any Category

- Any Class
- Any Instance

5 • Any Alarm

- Any Cmd

[0040] The table in Fig. 8 summarizes which filter criteria are valid for each event category:

[0041] The event distributor processes filters by examining the specified category and AND'ing together valid criteria. Clients may simulate OR operations by registering multiple filters.

[0042] The EvtDist service object implements the following client interfaces:

- RegisterFilter
A client uses this method to register an event filter. A filter identifier is returned.
- CancelFilter
A client invokes this method to remove a specified event filter, using the filter id returned from the associated registration.

[0043] Note that each method requires a client session application identifier to validate user access.

[0044] Refer to the description of interfaces EvtDist & EventCB in the attachment for additional details.

25 Alarm Manager

[0045] Alarm information is reported as a sequence of AlarmData structures which contain:

- The ClassCode of a managed object which defines a network-element specific alarm record.
Note that in the first release of the EMS, only one network element active alarm table is defined (ApActiveAlarms).
- A sequence of alarm records, each of which contains an alarm instance identifier and sequence of attribute code-value pairs.
Client applications may request a copy of all active alarms filtered on any combination of the following:

- Network element
- Maintenance unit
- Alarm level

[0046] Similar to the interfaces provided by the event distributor, out-of-band values may be used to represent wild-cards.

[0047] Since managed object instance information may not be available at the time an alarm is reported, the actual alarm filter criteria are specified in terms of logical identifiers. Logical ids are integer values which represent the logical numbers of devices and interfaces (e.g. AP 4). The correlation between logical ids and managed object instance identifiers is provided in the configuration information made available by each managed object service object, and through the utility method getInst. Refer to the section on Managed Objects for additional details.

[0048] The following AlarmManager client interfaces are written specifically for the Active Alarm List application:

- RequestAlarms
A client invokes this method to register a filter for active alarms.
- ChangeFilter
A client may invoke this method to change filter criteria.
- RefreshAlarms
A client may invoke this method to refresh the active alarm list.

- CancelAlarms

A client should invoke this method to de-register a filter.

5 [0049] All operations except for de-registration return all active alarms filtered on the specified criteria. Also, each of these methods requires a valid client session application identifier to validate user access, and to keep track of the additional server resources which may be utilized while each client is active.

[0050] The following AlarmManager interface may be used by any client application (e.g. CLI):

- opAlarm

10 Through client implementations of event callbacks used to process command acknowledgements and responses (the same EventCB reference may be used in both cases), this method returns either a list of all active alarms in the system or just those associated with a target network element.

15 [0051] Refer to the description of interfaces AlarmManager, AlarmCB & EventCB in the attachment for additional details.

Exceptions

[0052] Exceptions are used for consistent and structured error handling in both the EM Server and Client.

20 [0053] The CORBA specification defines many system exceptions:

- BAD_PARAM
- INV_OBJREF
- NO_PERMISSION
- BAD_OPERATION
- 30 • OBJ_ADAPTER
- ♦♦♦

35 [0054] Refer to "The Common Object Request Broker: Architecture and Specification" for an exhaustive list of mnemonics and the associated exception descriptions.

[0055] Vendor-specific object request broker exceptions are also defined (using the Minor identifier of the SystemException):

- NO_IT_DAEMON_PORT
- 40 • LICENCE_EXPIRED
- ♦♦♦

45 [0056] Currently, the EMS uses Iona's Orbix product. Refer to the "Orbix 2.3c Reference Guide" for an exhaustive list of mnemonics and the associated exception descriptions.

[0057] In most cases, exceptions will be treated as fatal errors by Client code resulting in application termination.

[0058] An interface 26-specific exception is also defined as an Exception Code containing one of the following values shown in Fig. 9.

50

55

APPENDIX A- Emapi.idl

```

5      #ifndef _EMAPI_IDL
6      #define _EMAPI_IDL

7      //
8      // File: Emapi.idl
9      //
10     // Description: CORBA IDL file for the Element Manager API. All client
11         visible interfaces that are common to all applications of the
12         EMS
13         are described here.
14         //

15     // interfaces which are referenced before they are defined
16     interface ConfigCB;
17     interface StatusCB;
18     interface NEConfigCB;
19     interface EventCB;
20     interface AlarmCB;

21     // some of the more commonly used ASN.1 standard types
22     typedef boolean          Asn1Boolean;
23     typedef long             Asn1Integer;
24     typedef unsigned long    Asn1UInteger;
25     typedef sequence<octet> Asn1Octet;
26     typedef unsigned long    Asn1Timeticks;
27     typedef unsigned long    Asn1Gauge;
28     typedef unsigned long    Asn1Counter;
29     typedef octet            Asn1IpAddress[4];
30     typedef octet            Asn1Null;
31     typedef sequence<unsigned long> Asn1Oid;

32     // logical identifier--one or more attributes of this type may be
33     // defined
34     // for any managed object to provide a logical identity (e.g.
35     // application
36     // processor number, RCS number)
37     typedef Asn1Integer        LogicalId;
38     typedef sequence<LogicalId> SeqLogicalId;

39     // definition of an object identifier
40     typedef long               ClassCode;
41     typedef unsigned long      InstId;

42     struct Oid {
43         ClassCode            classId;
44         InstId               instId;
45     };

46     // reserved class id
47     const ClassCode          AnyClass      = -1;

48     // reserved instance id's
49     const InstId             NullInstance = 0;
50     const InstId             AnyInstance  = 1;
51     const InstId             MaxRsvdInst = 1;

```

```

// instance id associated with singleton objects (e.g. System)--note
5   that
// this is the first id normally assigned
const InstId           SingletonInst  = MaxRsvdInst + 1;

// reserved logical id
const LogicalId        AnyLogicalId = -1;

10  // application identifier
typedef long             AppId;
const AppId              NullAppId    = -1;
const AppId              AnyAppId    = -2;

15  // application/session identifier
struct SessionAppId {
    InstId           session;
    AppId            app;
};

20  // maximum number of applications active per single user session
const short              MaxSessionApps = 10;

// command identifier
25  typedef long             CmdSeqNo;
const CmdSeqNo           AnyCmd        = -1;
const CmdSeqNo           InvalidCmd   = -1;
30  struct CmdId {
    InstId           session;
    CmdSeqNo         seqNo;
};

35  // element manager base application programming interface exception
ccdes
enum EmapiExceptionCode {
    EM_INVALID_USER,
    EM_UNKNOWN_HOST,
    EM_TOO_MANY_USER_SESSIONS,
    EM_TOO_MANY_APPLICATIONS,
    EM_INVALID_SESSION_ID,
    EM_INVALID_APP_ID,
    EM_INVALID_INST_ID,
    EM_INVALID_NE_ID,
    EM_INVALID_MO_ID,
    EM_INVALID_ATTR_CODE,
    EM_NO_MATCHING_INST,
    EM_INVALID_FILTER,
    EM_INVALID_FILTER_ID,
    EM_NE_ISOLATED,
    EM_INTERNAL_ERROR,
    EM_INVALID_OPERATION,
    EM_ACCESS_DENIED,
    EM_VERSION_MISMATCH,
    EM_LOST_RESOURCE,
    EM_INVALID_KEY,
    EM_INVALID_CATEGORY
};

```

```

1;

// element manager exception
exception EmapiException { EmapiExceptionCode errCode; };

5
// Access permissions:
// For now, permission is granted on a network element basis:
// on any kind of access
// on status information access
// on maintenance operation access
10
// The network element basis can be:
// on any network element
// Oid: {AnyClass,AnyInstance}
// on any instance of a particular network element type
// Oid: {<class>,AnyInstance}
15
// on a specific instance of a particular network element
type
// Oid: {<class>,<instance>}

20
// Kinds of Access (set to be manageable by using a bit map/mask):
typedef Asn1Integer AccessType;

25
const AccessType AnyAccess = -1;
const AccessType NoAccess = 0;
const AccessType StatusAccess = 1;
const AccessType OperationAccess = 2;
//const AccessType NextAccess = 4;
//const AccessType AfterThat = 8
//const AccessType AfterThat = 16
// ...and so on, in powers of 2

30
// AccessPermission structure:
// accessible TRUE or FALSE for this type of access
// accessType type of access
// oid network element involved in this type of
access

35
struct AccessPermission {
    boolean accessible;
    AccessType accessType;
    Oid oid;
};

40
// Application registration results in the client's receiving a
sequence of
// AccessPermission blocks.

45
typedef sequence<AccessPermission> SeqAccessPermission;

struct AccessPermissionList {
    SessionAppId sessionAppId;
    SeqAccessPermission seqAccessPermission;
};

50
// user session
interface UserSession {
    // Null Session
    const InstId NullSession = 0;
};

55

```

```

1           // maximum number of sessions per system
2           const short          MaxUserSessions      = 32;
3
4           // heartbeat period in seconds (Applications should use this
5           // value to
6           // time their heartbeats with the server.)
7           const long           HeartbeatPeriod      = 5;
8
9           // client.application registration: must be called once per
10          // application.
11          //
12          // INPUTS:
13          //     applicationName: determined by client application
14          //     host: name of host on which client is running
15          //     user: login id
16          //     passwd: user's password
17          // RETURNS:
18          //     unique id for this session's application
19          //     plus a sequence of AccessPermission blocks
20          //     showing what is accessible from this application
21          // NOTE: The client's responsibility to delete the returned
22          pointer
23          //
24          // THROWS:
25          //     EM_TOO_MANY_SESSIONS
26          //     EM_TOO_MANY_APPLICATIONS
27          AccessPermissionList startApplication(
28              in string host,
29              in string user,
30              in string passwd,
31              in string applicationName)
32              raises(EmapiException);

33          // client deregistration -- entire session
34          //
35          // INPUT: a session application id
36          //
37          // THROWS:
38          //     EM_INVALID_SESSION_ID
39          void stop(in InstId sessionId) raises(EmapiException);

40          // client deregistration -- single application
41          //
42          // INPUT: a session application id
43          //
44          // THROWS:
45          //     EM_INVALID_SESSION_ID
46          //     EM_INVALID_APPLICATION_ID
47          void stopApplication(in SessionAppId id)
48          raises(EmapiException);

49          // client heartbeat
50          //
51          // INPUT: session application id
52          //
53          // THROWS:
54          //     EM_INVALID_SESSION_ID
55          //     EM_INVALID_APP_ID
56          //     EM_INTERNAL_ERROR

```

```

    void heartbeat(in SessionAppId id) raises(EmapiException);
}

5
// Managed Object definition. The Managed Object (MO) describes
// definitions
// and operations that are common to all application specific managed
// objects in the system.
//
10 interface MO {
    // All attributes of an MO are identified by integer constant
    // codes called AttrCodes.
    typedef long AttrCode;
    const AttrCode NullAttrCode = -1;

15
    //
    // The following attribute codes are reserved and are used
    // by the MO implementation. Clients should always use the
    // attribute codes found in the application specific managed
    // object definition (e.g. ApModule.idl) and not these.
    //
20    const AttrCode moInstanceCode = 0;
    const AttrCode neInstanceCode = 1;
    const AttrCode LastReservedCode= 1;

25
    // clients register for updates by specifying a sequence of
    // attribute codes
    typedef sequence<AttrCode> SeqAttrCode;

    // The attribute value is a discriminated union of scalars
    // All basic types currently supported by the EMS are
30    described
        // here.
        typedef long AttrType;
        const AttrType ValInstId = 1;
        const AttrType ValAsn1Boolean = 2;
        const AttrType ValAsn1Integer = 3;
        const AttrType ValAsn1UInteger = 4;
        const AttrType ValAsn1Octet = 5;
        const AttrType ValAsn1Timeticks = 6;
        const AttrType ValAsn1Gauge = 7;
        const AttrType ValAsn1Counter = 8;
        const AttrType ValAsn1IpAddress = 9;
        const AttrType ValAsn1Null = 10;
        const AttrType ValAsn1Oid = 11;
        const AttrType ValLogicalId = 12;

35
        //
40        // The following union defines all possible attribute types
        // in the EMS.
        //
45        union AttrValue switch(AttrType) {
            case ValInstId:     InstId      instId;
            case ValAsn1Boolean: Asn1Boolean asn1Boolean;
            case ValAsn1Integer: Asn1Integer asn1Integer;
            case ValAsn1UInteger: Asn1UInteger asn1UInteger;
            case ValAsn1Octet:   Asn1Octet  asn1Octet;
            case ValAsn1Timeticks: Asn1Timeticks asn1Timeticks;
            case ValAsn1Gauge:   Asn1Gauge  asn1Gauge;
        }
50
55

```

```

5           case ValAsn1Counter:      Asn1Counter      asn1Counter;
6           case ValAsn1IpAddress:   Asn1IpAddress    asn1IpAddress;
7           case ValAsn1Null:        Asn1Null         asn1Null;
8           case ValAsn1Oid:        Asn1Oid          asn1Oid;
9           case ValLogicalId:     LogicalId        logicalId;
10          };

11          //
12          // each set of updates on a managed object is delivered
13          // as a sequence of attribute-value pairs (SeqAttrCodeValue).
14          //
15          struct AttrCodeValue {
16              AttrCode          code;
17              AttrValue         value;
18          };
19          typedef sequence<AttrCodeValue> SeqAttrCodeValue;
20          //
21          // The getAttrInfo method of the MO returns a sequence of
22          // information that describes each available attribute.
23          //
24          struct AttrInfo {
25              AttrCode          code;
26              AttrType         type; // note: currently not
27              populated
28                  Asn1Octet       name;
29          };
30          //
31          // Config update notifications (via the deliverConfig method
32          // in the
33          // ConfigCB) can take two forms.
34          //      1) One or more managed object instances have been
35          //          created.           (CONFIG_CREATE).
36          //      2) One or more managed object instances have been
37          //          deleted.          (CONFIG_DELETE).
38          //
39          enum ConfigNotifyType {
40              CONFIG_CREATE, // a new MO instance has been created
41              CONFIG_DELETE // an existing MO instance has been
42          deleted.
43          };

44          //
45          // Status update notifications (via the deliverStatus method
46          // in the
47          // StatusCB) can take two forms.
48          //      1) An initial update (STATUS_INIT) that is returned
49          //          either
50          //              as a result of a getStatus or as the initial status
51          //              from
52          //                  a startUpdate request.
53          //      2) An incremental update (STATUS_CHANGE) representing a
54          //          change
55

```

```

      // to one or more of the attributes that the client
5    registered for.
      //
      enum StatusNotifyType {
          STATUS_INIT,      // represents initial status update of
all           // requested attributes (e.g.
startUpdate)      STATUS_CHANGE // represents an incremental status
10      update           // (contains only attributes that have
                           // changed).
      };

15      //
      // each config update notification will be delivered as a
sequence of:
      //
      // network element instance id
      // managed object key list (sequence of attr-
20      value pairs)
      //
      // managed object instance id
      struct ConfigData {
          InstId           neInst;
          SeqAttrCodeValue keyList;
          InstId           moInst;
      };

25      typedef sequence<ConfigData>      SeqConfigData;

      //
30      // viewConfig() - obtain EMS view of the current managed
object
      // configuration for a specified network element
instance.
      // The special value AnyInstance may be used to obtain
      // configuration information for all network elements
35      known
      // to the EMS.
      // INPUTS:
      // sessionAppId - client session/application identifier.
This is
      // used to validate client permission to
40      perform
      // the operation.
      // neInst - specific NE instance identifier, or
AnyInstance
      // to get config for all NE instances.
      // RETURNS:
      // A sequence of configuration information (sequence
length
      // is proportional to the number of configured MO
instances).
      // CALLER MUST DELETE RETURNED MEMORY.
50      //
      // THROWS:
      // EM_INVALID_SESSION_ID
      // EM_INVALID_APP_ID
      // EM_INVALID_INST_ID
55      //

```

```

SeqConfigData viewConfig(in SessionAppId      sessionAppId,
                        in InstId            neInst)
                        raises(EmapiException);

5
  // notifyConfig() - Client registration for managed object
  // configuration information for specified network
element
  // instance (or the AnyInstance to register for
10 notifications
  // on all network element instances)... The current
snapshot
  // of configuration is returned (like viewConfig) and all
  // subsequent configuration changes result in an
15 invocation of
  // the specified callback.

  // INPUTS:
  // sessionAppId - client session/application identifier.
This is
20
  // used to validate client permission to
perform
  // the operation.
  // neInst - specific NE instance identifier, or
AnyInstance
25
  // to get config for all NE instances.
  // callback - client callback (implements the
ConfigCB
  // interface) that will be invoked for
all
  // config changes.

30
  // RETURNS:
  // A sequence of configuration information (sequence
length
  // is proportional to the number of configured MO
instances).
  // CALLER MUST DELETE RETURNED MEMORY.

35
  // THROWS:
  // EM_INVALID_SESSION_ID
  // EM_INVALID_APP_ID
  // EM_INVALID_INST_ID
40
  SeqConfigData notifyConfig(in SessionAppId      sessionAppId,
                            in InstId            neInst,
                            in ConfigCB          callback)
                            raises(EmapiException);

45
  // -// cancelNotify() - Cancel all requests for configuration
change
  // notifications associated with the specified client
application.
50
  // INPUTS:
  // sessionAppId - client session/application identifier.
This is
  // used to validate client permission to
perform
55

```

```

10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55

```

```

//                               the operation.
// RETURNS:
//      void
//
// THROWS:
//      EM_INVALID_SESSION_ID
//      EM_INVALID_APP_ID
//
void cancelNotify(in SessionAppId sessionAppId)
raises(EmapiException);

//
// getPersistent() - Obtain the attribute codes for all
persistent
//      attributes maintained by this managed object.
Persistent
//      attributes are those that the EMS keeps the current
value
//      regardless of client registrations. Other attributes
are
//      polled for only when a client makes a request or
registers
//      for update notifications.
// INPUTS:
//      sessionAppId - client session/application identifier.
This is
//      used to validate client permission to
perform
//      the operation.
// RETURNS:
//      a sequence of attribute codes representing the
persistent
//      attributes.
//      CALLER MUST DELETE RETURNED MEMORY.
//
// THROWS:
//      EM_INVALID_SESSION_ID
//      EM_INVALID_APP_ID
//
SeqAttrCode getPersistent(in SessionAppId sessionAppId)
raises(EmapiException);

//
// getAttrSpec() - Return identification of attributes that
are
//      defined for the specific managed object instance. A
sequence
//      containing the attribute name (an OCTET representing
the
//      ASCII string name) and the associated attribute code
//      is returned.
//
// INPUTS:
//      sessionAppId - client session/application identifier.
This is
//      used to validate client permission to
perform
//      the operation.
// RETURNS:

```

```

5           // A sequence of attribute info representing the
           // attribute names and codes.
           // CALLER MUST DELETE RETURNED MEMORY.
           //
           // THROWS:
           //   EM_INVALID_SESSION_ID
           //   EM_INVALID_APP_ID
           //
10          SeqAttrInfo getAttrSpec(in SessionAppId sessionAppId)
           raises(EmapiException);

           //
           // getKeySpec() - Return identification of key attributes. A
           sequence
           //          of attribute codes representing the keys is returned.
15          Note
           //          that the sequence will be in the order defined in the
           //          MOview.
           //
           // INPUTS:
           //          sessionAppId - client session/application identifier.
20          This is
           //          used to validate client permission to
           perform
           //          the operation.
           //
           // RETURNS:
           //          A sequence of attribute codes.
           //          CALLER MUST DELETE RETURNED MEMORY.
           //
25          // THROWS:
           //   EM_INVALID_SESSION_ID
           //   EM_INVALID_APP_ID
           //
           SeqAttrCode      getKeySpec(in SessionAppId sessionAppId)
           raises(EmapiException);

           //
           // viewStatus() - obtain EMS "view" of the values of the
           specified
           //          persistent attributes.
           // INPUTS:
           //          sessionAppId - client session/application identifier.
40          This is
           //          used to validate client permission to
           perform
           //          the operation.
           //          instId      - MO instance. This specifies the MO
           whose
           //          attributes are to be examined.
           //          attrList    - sequence of attribute codes that are to
           be
           //          viewed.
           // RETURNS:
           //          a sequence of attribute code/value pairs representing
           the
           //          current view of the attribute values.
           //          CALLER MUST DELETE RETURNED MEMORY.
           //
55

```

```

5           // THROWS:
10          //   EM_INVALID_SESSION_ID
11          //   EM_INVALID_APP_ID
12          //   EM_INVALID_INST_ID
13          //   EM_INVALID_MO_ID
14          //   EM_INVALID_ATTR_CODE
15
16          // SeqAttrCodeValue viewStatus(in SessionAppId      sessionAppId,
17                           in InstId           instId,
18                           in SeqAttrCode      attrList)
19                           raises(EmapiException);

20
21          // // getStatus() - request for a snapshot of current status
22          //   information. This differs from viewStatus in that
23          attrList
24          //   may specify any managed object attribute codes, and
25          the
26          //   information is returned via client callback. The
27          callback is
28          //   used because the request may require a get request to
29          the
30          //   network element.
31
32          // INPUTS:
33          //   sessionAppId - client session/application identifier.
34          This is
35          //   used to validate client permission to
36          perform
37          //   the operation.
38          //   instId      - MO instance. This specifies the MO
39          //   attributes are to be examined.
40          //   attrList    - sequence of attribute codes that are to
41          be
42          //   retrieved.
43
44          // RETURNS:
45          //   void
46
47          // THROWS:
48          //   EM_INVALID_SESSION_ID
49          //   EM_INVALID_APP_ID
50          //   EM_INVALID_INST_ID
51          //   EM_INVALID_MO_ID
52          //   EM_INVALID_ATTR_CODE
53
54          void getStatus(in SessionAppId  sessionAppId,
55                           in InstId           instId,
56                           in SeqAttrCode      attrList,
57                           in StatusCB         callback)
58                           raises(EmapiException);

59
60          // // startUpdate() - client registration for notifications of
61          //   any updates to the values of the specified set of
62          attributes.
63          //   An initial snapshot of all requested attributes is
64          //   delivered first (type=STATUS_INIT) followed by
65          notifications

```

55

```

      //      of only those attributes that have changed
5     (type=STATUS_CHANGE)
      //      Note that this method may result in polling to the
      network
      //      element, but only if the attributes are not persistent
      and
      //      no other clients have issued a startUpdate for the
      attributes (only a single poll is used for all
      registered
      //      clients).

      //
      // INPUTS:
      //      sessionAppId - client session/application identifier.
      This is
      //      used to validate client permission to
      perform
      //      the operation.
      //      instId      - MO instance. This specifies the MO
      whose attributes are to be examined.
      //      attrList    - sequence of attribute codes that are to
      be
      //      monitored.

      // RETURNS:
      //      void
      //

      // THROWS:
      //      EM_INVALID_SESSION_ID
      //      EM_INVALID_APP_ID
      //      EM_INVALID_INST_ID
      //      EM_INVALID_MO_ID
      //      EM_INVALID_ATTR_CODE
      //

      void startUpdate(in SessionAppId sessionAppId,
                      in InstId instId,
                      in SeqAttrCode attrList,
                      in StatusCB callback)
      raises(EmapiException);

      //
      // stopUpdate() - client deregistration for status updates.
      Cancel
      //      all monitoring activity associated with the specified
      client
      //      application. Note that the client may have issued a
      number
      //      of startUpdate requests for different instances, but
      this
      //      single call to stopUpdate will cancel all of those
      requests.
      //
      // INPUTS:
      //      sessionAppId - client session/application identifier.
      This is
      //      used to validate client permission to
      perform
      //      the operation.

      // RETURNS:
      //      void
      //

```

```

// THROWS:
//      EM_INVALID_SESSION_ID
//      EM_INVALID_APP_ID
5      void stopUpdate(in SessionAppId sessionAppId)
raises(EmapiException);
//
//      // getInst() - return the instance identifier associated with
the
10     //      specified keys.
//
//      // INPUTS:
//      sessionAppId - client session/application identifier.
This is
15     //      used to validate client permission to
perform
//      the operation.
//      neInst      - the network element instance identifier
of the
20     //      of the
//      NE that contains the managed object
instance
//      specified by the keys.
//
//      // RETURNS:
//      the InstId of the managed object, or NullInstance if
not found.
25     //
//      // THROWS:
//      EM_INVALID_SESSION_ID
//      EM_INVALID_APP_ID
//      EM_INVALID_INST_ID
//
30     InstId getInst(in SessionAppId           sessionAppId,
                  in InstId                 neInst,
                  in SeqAttrCodeValue      keyValues)
raises(EmapiException);
};

35
//
// client callback for managed object configuration reporting
//
40     interface ConfigCB {
        oneway void deliverConfig(in ClassCode           classId,
                                in MO::ConfigNotifyType type,
                                in MO::SeqConfigData  config);
};

45
//
// client callback for managed object status reporting
//
50     interface StatusCB {
        oneway void deliverStatus(in Oid           oid,
                                in MO::StatusNotifyType type,
                                in MO::SeqAttrCodeValue
attrValList);
};

// network element level managed object
55     interface NEMO : MO {

```

```

// attribute code reserved for boolean network element
// isolation indication.
const AttrCode IsolatedCode = 1;

5
//
// each network level configuration update notification is
delivered
// as a sequence of:
//           network element instance id
//           network element logical id
10
struct NEconfigData {
    InstId           instId;
    LogicalId        logicalId;
};

15
typedef sequence<NEconfigData> SeqNEconfigData;

// ...
// Sequence of containment information describing all
// contained managed objects.
//
20
struct ContainmentInfo {
    ClassCode        classCode;
    MO              moRef;
    Asn1Octet       name;
};

25
typedef sequence<ContainmentInfo> SeqContainmentInfo;

//
// viewNEconfig() - obtain EMS view of current network element
// configuration
30
//
// INPUTS:
//           sessionAppId - client session/application identifier.
This is
//           used to validate client permission to
perform
//           the operation.
//
// RETURNS:
//           Sequence of network element configuration information.
//           CALLER MUST DELETE RETURNED MEMORY.
//
40
// THROWS:
//           EM_INVALID_SESSION_ID
//           EM_INVALID_APP_ID
//
SeqNEconfigData viewNEconfig(in SessionAppId sessionAppId)
                     raises(EmapiException);

45
//
// notifyNEconfig() - client registration for network level
// managed object configuration. An initial snapshot of
// the network element level configuration is returned.
// All subsequent changes to the network element
// configuration are delivered via the specified callback.
//
50
// INPUTS:
//           sessionAppId - client session/application identifier.
This is

```

```

      //
      // used to validate client permission to
      // perform
      // the operation.
      // callback - callback object that is to receive
      // delivery
      // of changes to configuration.
      // RETURNS:
      // Sequence of network element configuration information.
      // CALLER MUST DELETE RETURNED MEMORY.
      //
      // THROWS:
      // EM_INVALID_SESSION_ID
      // EM_INVALID_APP_ID
      //
15     SeqNEconfigData notifyNEconfig(in SessionAppId sessionAppId,
      //                                     in NEconfigCB callback)
      //                                     raises(EmapiException);

      //
20     // cancelNEnotify() - cancel request for NE configuration
      // change notifications
      //
      // INPUTS:
      // sessionAppId - client session/application identifier.
      This is
      // used to validate client permission to
      // perform
      // the operation.
      // RETURNS:
      // void
      //
30     // THROWS:
      // EM_INVALID_SESSION_ID
      // EM_INVALID_APP_ID
      //
      void cancelNEnotify(in SessionAppId sessionAppId)
      // raises(EmapiException);

      //
40     // getNEInst() - return the network element instance
      // identifier
      // associated with the specified NE logical identifier.
      //
      // INPUTS:
      // sessionAppId - client session/application identifier.
      This is
      // used to validate client permission to
      // perform
      // the operation.
      // logicalId - the logical identifier (integer) of
      // the network element.
      // RETURNS:
      // the InstId of the network element instance, or
50     NullInstance
      // if not found.
      //
      // THROWS:
      // EM_INVALID_SESSION_ID
      // EM_INVALID_APP_ID
      //
55

```

```

5
// InstId getNEinst(in SessionAppId           sessionAppId,
5      in LogicalId           logicalId)
5      raises(EmapiException);

10
// // getLogicalId() - return the NE logical identifier
10      // associated with the specified instance identifier.
10
10      // INPUTS:
10      //      neInst      - the instance identifier of the network
10      element.
10
10      // RETURNS:
10      //      the logical identifier of the network element
10      instance,
10      //      or 0 if not found.
10
10      Asn1Integer getLogicalId(in InstId neInst);

20
// // getContainment() - return sequence with containment
20      information
20      //      for this network element. The sequence returned
20      contains
20      //      the name, class code and a pointer to the associated
20      service
20      //      class object.
20
20      // INPUTS:
20      //      sessionAppId - client session/application identifier.
20
20      This is
20      //      used to validate client permission to
20      perform
20      //      the operation.
20
20      // RETURNS:
20      //      a sequence of containment information describing the
20      contained
20      //      managed objects.
20      //      CALLER MUST DELETE RETURNED MEMORY.
20
20      // THROWS:
20      //      EM_INVALID_SESSION_ID
20      //      EM_INVALID_APP_ID
20
20      //      SeqContainmentInfo getContainment(in SessionAppId
20      sessionAppId)
20
20      //      raises(EmapiException);

45
};

// client callback for network element level managed object
45      configuration
45      // reporting
45      interface NEconfigCB {
45          oneway void deliverNEconfig(in ClassCode           classId,
45                                     in NEMO::ConfigNotifyType type,
45                                     in NEMO::SeqNEconfigData config);
45
55

```

```

5 // typedefs for acknowledgement value & alarm level
6     typedef Asn1Integer          AckValue;
7     typedef Asn1Integer          AlarmLevel;

10 // no alarm filtering conjunctions are permitted, but we do allow a
11 // single
12 // filter to extract all alarmed notifications by defining an "out-of-
13 // band"
14 // alarm level
15 const   AlarmLevel           AnyAlarm      = -1;

19 // typedef for event data structure--note that instance information
20 // may no
21 // longer be valid when an event is processed, so key information
22 // should
23 // always be available in the attribute-value list
24 struct EventData {
25     ClassCode                  classCode;
26     MO::SeqAttrCodeValue       seqVal;
27 };
28 typedef sequence<EventData>      SeqEventData;

32 // event distributor interface description
33 interface EvtDist {
34     // time of event as reported by EMS
35     typedef long                EventTime;

39     // categories of events
40     enum Category {
41         ANY_EVENT,
42         ALARM_CLEAR,
43         ALARM_SET,
44         CMD_ACK,
45         CMD_RESP,
46         CONFIG_CHANGE,
47         INFO_MSG,
48         INIT_MSG,
49         STATE_CHANGE,
50         NUM_CATEGORIES
51     };
52
53     // event header (not all members valid for all categories)
54     struct Header {
55         EventTime                eventTime;
56         Category                 category;
57         Oid                      networkElemId;
58         AlarmLevel               networkElemAlm;
59         Oid                      maintUnitId;
60         AlarmLevel               maintUnitIdAlm;
61         CmdId                   cmdId;
62     };
63
64     // event filter (not all members valid for all categories)
65     struct Filter {
66         Category                 category;
67         Oid                      networkElemId;
68         AlarmLevel               networkElemAlm;
69         Oid                      maintUnitId;
70     };
71
72     // event header (not all members valid for all categories)
73     struct Header {
74         EventTime                eventTime;
75         Category                 category;
76         Oid                      networkElemId;
77         AlarmLevel               networkElemAlm;
78         Oid                      maintUnitId;
79     };
80
81     // event filter (not all members valid for all categories)
82     struct Filter {
83         Category                 category;
84         Oid                      networkElemId;
85         AlarmLevel               networkElemAlm;
86         Oid                      maintUnitId;
87     };
88
89     // event header (not all members valid for all categories)
90     struct Header {
91         EventTime                eventTime;
92         Category                 category;
93         Oid                      networkElemId;
94         AlarmLevel               networkElemAlm;
95         Oid                      maintUnitId;
96     };
97
98     // event filter (not all members valid for all categories)
99     struct Filter {
100        Category                category;
101        Oid                     networkElemId;
102        AlarmLevel              networkElemAlm;
103        Oid                     maintUnitId;
104    };
105
106    // event header (not all members valid for all categories)
107    struct Header {
108        EventTime               eventTime;
109        Category                category;
110        Oid                     networkElemId;
111        AlarmLevel              networkElemAlm;
112        Oid                     maintUnitId;
113    };
114
115    // event filter (not all members valid for all categories)
116    struct Filter {
117        Category                category;
118        Oid                     networkElemId;
119        AlarmLevel              networkElemAlm;
120        Oid                     maintUnitId;
121    };
122
123    // event header (not all members valid for all categories)
124    struct Header {
125        EventTime               eventTime;
126        Category                category;
127        Oid                     networkElemId;
128        AlarmLevel              networkElemAlm;
129        Oid                     maintUnitId;
130    };
131
132    // event filter (not all members valid for all categories)
133    struct Filter {
134        Category                category;
135        Oid                     networkElemId;
136        AlarmLevel              networkElemAlm;
137        Oid                     maintUnitId;
138    };
139
140    // event header (not all members valid for all categories)
141    struct Header {
142        EventTime               eventTime;
143        Category                category;
144        Oid                     networkElemId;
145        AlarmLevel              networkElemAlm;
146        Oid                     maintUnitId;
147    };
148
149    // event filter (not all members valid for all categories)
150    struct Filter {
151        Category                category;
152        Oid                     networkElemId;
153        AlarmLevel              networkElemAlm;
154        Oid                     maintUnitId;
155    };
156
157    // event header (not all members valid for all categories)
158    struct Header {
159        EventTime               eventTime;
160        Category                category;
161        Oid                     networkElemId;
162        AlarmLevel              networkElemAlm;
163        Oid                     maintUnitId;
164    };
165
166    // event filter (not all members valid for all categories)
167    struct Filter {
168        Category                category;
169        Oid                     networkElemId;
170        AlarmLevel              networkElemAlm;
171        Oid                     maintUnitId;
172    };
173
174    // event header (not all members valid for all categories)
175    struct Header {
176        EventTime               eventTime;
177        Category                category;
178        Oid                     networkElemId;
179        AlarmLevel              networkElemAlm;
180        Oid                     maintUnitId;
181    };
182
183    // event filter (not all members valid for all categories)
184    struct Filter {
185        Category                category;
186        Oid                     networkElemId;
187        AlarmLevel              networkElemAlm;
188        Oid                     maintUnitId;
189    };
190
191    // event header (not all members valid for all categories)
192    struct Header {
193        EventTime               eventTime;
194        Category                category;
195        Oid                     networkElemId;
196        AlarmLevel              networkElemAlm;
197        Oid                     maintUnitId;
198    };
199
200    // event filter (not all members valid for all categories)
201    struct Filter {
202        Category                category;
203        Oid                     networkElemId;
204        AlarmLevel              networkElemAlm;
205        Oid                     maintUnitId;
206    };
207
208    // event header (not all members valid for all categories)
209    struct Header {
210        EventTime               eventTime;
211        Category                category;
212        Oid                     networkElemId;
213        AlarmLevel              networkElemAlm;
214        Oid                     maintUnitId;
215    };
216
217    // event filter (not all members valid for all categories)
218    struct Filter {
219        Category                category;
220        Oid                     networkElemId;
221        AlarmLevel              networkElemAlm;
222        Oid                     maintUnitId;
223    };
224
225    // event header (not all members valid for all categories)
226    struct Header {
227        EventTime               eventTime;
228        Category                category;
229        Oid                     networkElemId;
230        AlarmLevel              networkElemAlm;
231        Oid                     maintUnitId;
232    };
233
234    // event filter (not all members valid for all categories)
235    struct Filter {
236        Category                category;
237        Oid                     networkElemId;
238        AlarmLevel              networkElemAlm;
239        Oid                     maintUnitId;
240    };
241
242    // event header (not all members valid for all categories)
243    struct Header {
244        EventTime               eventTime;
245        Category                category;
246        Oid                     networkElemId;
247        AlarmLevel              networkElemAlm;
248        Oid                     maintUnitId;
249    };
250
251    // event filter (not all members valid for all categories)
252    struct Filter {
253        Category                category;
254        Oid                     networkElemId;
255        AlarmLevel              networkElemAlm;
256        Oid                     maintUnitId;
257    };
258
259    // event header (not all members valid for all categories)
260    struct Header {
261        EventTime               eventTime;
262        Category                category;
263        Oid                     networkElemId;
264        AlarmLevel              networkElemAlm;
265        Oid                     maintUnitId;
266    };
267
268    // event filter (not all members valid for all categories)
269    struct Filter {
270        Category                category;
271        Oid                     networkElemId;
272        AlarmLevel              networkElemAlm;
273        Oid                     maintUnitId;
274    };
275
276    // event header (not all members valid for all categories)
277    struct Header {
278        EventTime               eventTime;
279        Category                category;
280        Oid                     networkElemId;
281        AlarmLevel              networkElemAlm;
282        Oid                     maintUnitId;
283    };
284
285    // event filter (not all members valid for all categories)
286    struct Filter {
287        Category                category;
288        Oid                     networkElemId;
289        AlarmLevel              networkElemAlm;
290        Oid                     maintUnitId;
291    };
292
293    // event header (not all members valid for all categories)
294    struct Header {
295        EventTime               eventTime;
296        Category                category;
297        Oid                     networkElemId;
298        AlarmLevel              networkElemAlm;
299        Oid                     maintUnitId;
300    };
301
302    // event filter (not all members valid for all categories)
303    struct Filter {
304        Category                category;
305        Oid                     networkElemId;
306        AlarmLevel              networkElemAlm;
307        Oid                     maintUnitId;
308    };
309
310    // event header (not all members valid for all categories)
311    struct Header {
312        EventTime               eventTime;
313        Category                category;
314        Oid                     networkElemId;
315        AlarmLevel              networkElemAlm;
316        Oid                     maintUnitId;
317    };
318
319    // event filter (not all members valid for all categories)
320    struct Filter {
321        Category                category;
322        Oid                     networkElemId;
323        AlarmLevel              networkElemAlm;
324        Oid                     maintUnitId;
325    };
326
327    // event header (not all members valid for all categories)
328    struct Header {
329        EventTime               eventTime;
330        Category                category;
331        Oid                     networkElemId;
332        AlarmLevel              networkElemAlm;
333        Oid                     maintUnitId;
334    };
335
336    // event filter (not all members valid for all categories)
337    struct Filter {
338        Category                category;
339        Oid                     networkElemId;
340        AlarmLevel              networkElemAlm;
341        Oid                     maintUnitId;
342    };
343
344    // event header (not all members valid for all categories)
345    struct Header {
346        EventTime               eventTime;
347        Category                category;
348        Oid                     networkElemId;
349        AlarmLevel              networkElemAlm;
350        Oid                     maintUnitId;
351    };
352
353    // event filter (not all members valid for all categories)
354    struct Filter {
355        Category                category;
356        Oid                     networkElemId;
357        AlarmLevel              networkElemAlm;
358        Oid                     maintUnitId;
359    };
360
361    // event header (not all members valid for all categories)
362    struct Header {
363        EventTime               eventTime;
364        Category                category;
365        Oid                     networkElemId;
366        AlarmLevel              networkElemAlm;
367        Oid                     maintUnitId;
368    };
369
370    // event filter (not all members valid for all categories)
371    struct Filter {
372        Category                category;
373        Oid                     networkElemId;
374        AlarmLevel              networkElemAlm;
375        Oid                     maintUnitId;
376    };
377
378    // event header (not all members valid for all categories)
379    struct Header {
380        EventTime               eventTime;
381        Category                category;
382        Oid                     networkElemId;
383        AlarmLevel              networkElemAlm;
384        Oid                     maintUnitId;
385    };
386
387    // event filter (not all members valid for all categories)
388    struct Filter {
389        Category                category;
390        Oid                     networkElemId;
391        AlarmLevel              networkElemAlm;
392        Oid                     maintUnitId;
393    };
394
395    // event header (not all members valid for all categories)
396    struct Header {
397        EventTime               eventTime;
398        Category                category;
399        Oid                     networkElemId;
400        AlarmLevel              networkElemAlm;
401        Oid                     maintUnitId;
402    };
403
404    // event filter (not all members valid for all categories)
405    struct Filter {
406        Category                category;
407        Oid                     networkElemId;
408        AlarmLevel              networkElemAlm;
409        Oid                     maintUnitId;
410    };
411
412    // event header (not all members valid for all categories)
413    struct Header {
414        EventTime               eventTime;
415        Category                category;
416        Oid                     networkElemId;
417        AlarmLevel              networkElemAlm;
418        Oid                     maintUnitId;
419    };
420
421    // event filter (not all members valid for all categories)
422    struct Filter {
423        Category                category;
424        Oid                     networkElemId;
425        AlarmLevel              networkElemAlm;
426        Oid                     maintUnitId;
427    };
428
429    // event header (not all members valid for all categories)
430    struct Header {
431        EventTime               eventTime;
432        Category                category;
433        Oid                     networkElemId;
434        AlarmLevel              networkElemAlm;
435        Oid                     maintUnitId;
436    };
437
438    // event filter (not all members valid for all categories)
439    struct Filter {
440        Category                category;
441        Oid                     networkElemId;
442        AlarmLevel              networkElemAlm;
443        Oid                     maintUnitId;
444    };
445
446    // event header (not all members valid for all categories)
447    struct Header {
448        EventTime               eventTime;
449        Category                category;
450        Oid                     networkElemId;
451        AlarmLevel              networkElemAlm;
452        Oid                     maintUnitId;
453    };
454
455    // event filter (not all members valid for all categories)
456    struct Filter {
457        Category                category;
458        Oid                     networkElemId;
459        AlarmLevel              networkElemAlm;
460        Oid                     maintUnitId;
461    };
462
463    // event header (not all members valid for all categories)
464    struct Header {
465        EventTime               eventTime;
466        Category                category;
467        Oid                     networkElemId;
468        AlarmLevel              networkElemAlm;
469        Oid                     maintUnitId;
470    };
471
472    // event filter (not all members valid for all categories)
473    struct Filter {
474        Category                category;
475        Oid                     networkElemId;
476        AlarmLevel              networkElemAlm;
477        Oid                     maintUnitId;
478    };
479
480    // event header (not all members valid for all categories)
481    struct Header {
482        EventTime               eventTime;
483        Category                category;
484        Oid                     networkElemId;
485        AlarmLevel              networkElemAlm;
486        Oid                     maintUnitId;
487    };
488
489    // event filter (not all members valid for all categories)
490    struct Filter {
491        Category                category;
492        Oid                     networkElemId;
493        AlarmLevel              networkElemAlm;
494        Oid                     maintUnitId;
495    };
496
497    // event header (not all members valid for all categories)
498    struct Header {
499        EventTime               eventTime;
500        Category                category;
501        Oid                     networkElemId;
502        AlarmLevel              networkElemAlm;
503        Oid                     maintUnitId;
504    };
505
506    // event filter (not all members valid for all categories)
507    struct Filter {
508        Category                category;
509        Oid                     networkElemId;
510        AlarmLevel              networkElemAlm;
511        Oid                     maintUnitId;
512    };
513
514    // event header (not all members valid for all categories)
515    struct Header {
516        EventTime               eventTime;
517        Category                category;
518        Oid                     networkElemId;
519        AlarmLevel              networkElemAlm;
520        Oid                     maintUnitId;
521    };
522
523    // event filter (not all members valid for all categories)
524    struct Filter {
525        Category                category;
526        Oid                     networkElemId;
527        AlarmLevel              networkElemAlm;
528        Oid                     maintUnitId;
529    };
530
531    // event header (not all members valid for all categories)
532    struct Header {
533        EventTime               eventTime;
534        Category                category;
535        Oid                     networkElemId;
536        AlarmLevel              networkElemAlm;
537        Oid                     maintUnitId;
538    };
539
540    // event filter (not all members valid for all categories)
541    struct Filter {
542        Category                category;
543        Oid                     networkElemId;
544        AlarmLevel              networkElemAlm;
545        Oid                     maintUnitId;
546    };
547
548    // event header (not all members valid for all categories)
549    struct Header {
550        EventTime               eventTime;
551        Category                category;
552        Oid                     networkElemId;
553        AlarmLevel              networkElemAlm;
554        Oid                     maintUnitId;
555    };
556
557    // event filter (not all members valid for all categories)
558    struct Filter {
559        Category                category;
560        Oid                     networkElemId;
561        AlarmLevel              networkElemAlm;
562        Oid                     maintUnitId;
563    };
564
565    // event header (not all members valid for all categories)
566    struct Header {
567        EventTime               eventTime;
568        Category                category;
569        Oid                     networkElemId;
570        AlarmLevel              networkElemAlm;
571        Oid                     maintUnitId;
572    };
573
574    // event filter (not all members valid for all categories)
575    struct Filter {
576        Category                category;
577        Oid                     networkElemId;
578        AlarmLevel              networkElemAlm;
579        Oid                     maintUnitId;
580    };
581
582    // event header (not all members valid for all categories)
583    struct Header {
584        EventTime               eventTime;
585        Category                category;
586        Oid                     networkElemId;
587        AlarmLevel              networkElemAlm;
588        Oid                     maintUnitId;
589    };
590
591    // event filter (not all members valid for all categories)
592    struct Filter {
593        Category                category;
594        Oid                     networkElemId;
595        AlarmLevel              networkElemAlm;
596        Oid                     maintUnitId;
597    };
598
599    // event header (not all members valid for all categories)
600    struct Header {
601        EventTime               eventTime;
602        Category                category;
603        Oid                     networkElemId;
604        AlarmLevel              networkElemAlm;
605        Oid                     maintUnitId;
606    };
607
608    // event filter (not all members valid for all categories)
609    struct Filter {
610        Category                category;
611        Oid                     networkElemId;
612        AlarmLevel              networkElemAlm;
613        Oid                     maintUnitId;
614    };
615
616    // event header (not all members valid for all categories)
617    struct Header {
618        EventTime               eventTime;
619        Category                category;
620        Oid                     networkElemId;
621        AlarmLevel              networkElemAlm;
622        Oid                     maintUnitId;
623    };
624
625    // event filter (not all members valid for all categories)
626    struct Filter {
627        Category                category;
628        Oid                     networkElemId;
629        AlarmLevel              networkElemAlm;
630        Oid                     maintUnitId;
631    };
632
633    // event header (not all members valid for all categories)
634    struct Header {
635        EventTime               eventTime;
636        Category                category;
637        Oid                     networkElemId;
638        AlarmLevel              networkElemAlm;
639        Oid                     maintUnitId;
640    };
641
642    // event filter (not all members valid for all categories)
643    struct Filter {
644        Category                category;
645        Oid                     networkElemId;
646        AlarmLevel              networkElemAlm;
647        Oid                     maintUnitId;
648    };
649
650    // event header (not all members valid for all categories)
651    struct Header {
652        EventTime               eventTime;
653        Category                category;
654        Oid                     networkElemId;
655        AlarmLevel              networkElemAlm;
656        Oid                     maintUnitId;
657    };
658
659    // event filter (not all members valid for all categories)
660    struct Filter {
661        Category                category;
662        Oid                     networkElemId;
663        AlarmLevel              networkElemAlm;
664        Oid                     maintUnitId;
665    };
666
667    // event header (not all members valid for all categories)
668    struct Header {
669        EventTime               eventTime;
670        Category                category;
671        Oid                     networkElemId;
672        AlarmLevel              networkElemAlm;
673        Oid                     maintUnitId;
674    };
675
676    // event filter (not all members valid for all categories)
677    struct Filter {
678        Category                category;
679        Oid                     networkElemId;
680        AlarmLevel              networkElemAlm;
681        Oid                     maintUnitId;
682    };
683
684    // event header (not all members valid for all categories)
685    struct Header {
686        EventTime               eventTime;
687        Category                category;
688        Oid                     networkElemId;
689        AlarmLevel              networkElemAlm;
690        Oid                     maintUnitId;
691    };
692
693    // event filter (not all members valid for all categories)
694    struct Filter {
695        Category                category;
696        Oid                     networkElemId;
697        AlarmLevel              networkElemAlm;
698        Oid                     maintUnitId;
699    };
700
701    // event header (not all members valid for all categories)
702    struct Header {
703        EventTime               eventTime;
704        Category                category;
705        Oid                     networkElemId;
706        AlarmLevel              networkElemAlm;
707        Oid                     maintUnitId;
708    };
709
710    // event filter (not all members valid for all categories)
711    struct Filter {
712        Category                category;
713        Oid                     networkElemId;
714        AlarmLevel              networkElemAlm;
715        Oid                     maintUnitId;
716    };
717
718    // event header (not all members valid for all categories)
719    struct Header {
720        EventTime               eventTime;
721        Category                category;
722        Oid                     networkElemId;
723        AlarmLevel              networkElemAlm;
724        Oid                     maintUnitId;
725    };
726
727    // event filter (not all members valid for all categories)
728    struct Filter {
729        Category                category;
730        Oid                     networkElemId;
731        AlarmLevel              networkElemAlm;
732        Oid                     maintUnitId;
733    };
734
735    // event header (not all members valid for all categories)
736    struct Header {
737        EventTime               eventTime;
738        Category                category;
739        Oid                     networkElemId;
740        AlarmLevel              networkElemAlm;
741        Oid                     maintUnitId;
742    };
743
744    // event filter (not all members valid for all categories)
745    struct Filter {
746        Category                category;
747        Oid                     networkElemId;
748        AlarmLevel              networkElemAlm;
749        Oid                     maintUnitId;
750    };
751
752    // event header (not all members valid for all categories)
753    struct Header {
754        EventTime               eventTime;
755        Category                category;
756        Oid                     networkElemId;
757        AlarmLevel              networkElemAlm;
758        Oid                     maintUnitId;
759    };
760
761    // event filter (not all members valid for all categories)
762    struct Filter {
763        Category                category;
764        Oid                     networkElemId;
765        AlarmLevel              networkElemAlm;
766        Oid                     maintUnitId;
767    };
768
769    // event header (not all members valid for all categories)
770    struct Header {
771        EventTime               eventTime;
772        Category                category;
773        Oid                     networkElemId;
774        AlarmLevel              networkElemAlm;
775        Oid                     maintUnitId;
776    };
777
778    // event filter (not all members valid for all categories)
779    struct Filter {
780        Category                category;
781        Oid                     networkElemId;
782        AlarmLevel              networkElemAlm;
783        Oid                     maintUnitId;
784    };
785
786    // event header (not all members valid for all categories)
787    struct Header {
788        EventTime               eventTime;
789        Category                category;
790        Oid                     networkElemId;
791        AlarmLevel              networkElemAlm;
792        Oid                     maintUnitId;
793    };
794
795    // event filter (not all members valid for all categories)
796    struct Filter {
797        Category                category;
798        Oid                     networkElemId;
799        AlarmLevel              networkElemAlm;
800        Oid                     maintUnitId;
801    };
802
803    // event header (not all members valid for all categories)
804    struct Header {
805        EventTime               eventTime;
806        Category                category;
807        Oid                     networkElemId;
808        AlarmLevel              networkElemAlm;
809        Oid                     maintUnitId;
810    };
811
812    // event filter (not all members valid for all categories)
813    struct Filter {
814        Category                category;
815        Oid                     networkElemId;
816        AlarmLevel              networkElemAlm;
817        Oid                     maintUnitId;
818    };
819
820    // event header (not all members valid for all categories)
821    struct Header {
822        EventTime               eventTime;
823        Category                category;
824        Oid                     networkElemId;
825        AlarmLevel              networkElemAlm;
826        Oid                     maintUnitId;
827    };
828
829    // event filter (not all members valid for all categories)
830    struct Filter {
831        Category                category;
832        Oid                     networkElemId;
833        AlarmLevel              networkElemAlm;
834        Oid                     maintUnitId;
835    };
836
837    // event header (not all members valid for all categories)
838    struct Header {
839        EventTime               eventTime;
840        Category                category;
841        Oid                     networkElemId;
842        AlarmLevel              networkElemAlm;
843        Oid                     maintUnitId;
844    };
845
846    // event filter (not all members valid for all categories)
847    struct Filter {
848        Category                category;
849        Oid                     networkElemId;
850        AlarmLevel              networkElemAlm;
851        Oid                     maintUnitId;
852    };
853
854    // event header (not all members valid for all categories)
855    struct Header {
856        EventTime               eventTime;
857        Category                category;
858        Oid                     networkElemId;
859        AlarmLevel              networkElemAlm;
860        Oid                     maintUnitId;
861    };
862
863    // event filter (not all members valid for all categories)
864    struct Filter {
865        Category                category;
866        Oid                     networkElemId;
867        AlarmLevel              networkElemAlm;
868        Oid                     maintUnitId;
869    };
870
871    // event header (not all members valid for all categories)
872    struct Header {
873        EventTime               eventTime;
874        Category                category;
875        Oid                     networkElemId;
876        AlarmLevel              networkElemAlm;
877        Oid                     maintUnitId;
878    };
879
880    // event filter (not all members valid for all categories)
881    struct Filter {
882        Category                category;
883        Oid                     networkElemId;
884        AlarmLevel              networkElemAlm;
885        Oid                     maintUnitId;
886    };
887
888    // event header (not all members valid for all categories)
889    struct Header {
890        EventTime               eventTime;
891        Category                category;
892        Oid                     networkElemId;
893        AlarmLevel              networkElemAlm;
894        Oid                     maintUnitId;
895    };
896
897    // event filter (not all members valid for all categories)
898    struct Filter {
899        Category                category;
900        Oid                     networkElemId;
901        AlarmLevel              networkElemAlm;
902        Oid                     maintUnitId;
903    };
904
905    // event header (not all members valid for all categories)
906    struct Header {
907        EventTime               eventTime;
908        Category                category;
909        Oid                     networkElemId;
910        AlarmLevel              networkElemAlm;
911        Oid                     maintUnitId;
912    };
913
914    // event filter (not all members valid for all categories)
915    struct Filter {
916        Category                category;
917        Oid                     networkElemId;
918        AlarmLevel              networkElemAlm;
919        Oid                     maintUnitId;
920    };
921
922    // event header (not all members valid for all categories)
923    struct Header {
924        EventTime               eventTime;
925        Category                category;
926        Oid                     networkElemId;
927        AlarmLevel              networkElemAlm;
928        Oid                     maintUnitId;
929    };
930
931    // event filter (not all members valid for all categories)
932    struct Filter {
933        Category                category;
934        Oid                     networkElemId;
935        AlarmLevel              networkElemAlm;
936        Oid                     maintUnitId;
937    };
938
939    // event header (not all members valid for all categories)
940    struct Header {
941        EventTime               eventTime;
942        Category                category;
943        Oid                     networkElemId;
944        AlarmLevel              networkElemAlm;
945        Oid                     maintUnitId;
946    };
947
948    // event filter (not all members valid for all categories)
949    struct Filter {
950        Category                category;
951        Oid                     networkElemId;
952        AlarmLevel              networkElemAlm;
953        Oid                     maintUnitId;
954    };
955
956    // event header (not all members valid for all categories)
957    struct Header {
958        EventTime               eventTime;
959        Category                category;
960        Oid                     networkElemId;
961        AlarmLevel              networkElemAlm;
962        Oid                     maintUnitId;
963    };
964
965    // event filter (not all members valid for all categories)
966    struct Filter {
967        Category                category;
968        Oid                     networkElemId;
969        AlarmLevel              networkElemAlm;
970        Oid                     maintUnitId;
971    };
972
973    // event header (not all members valid for all categories)
974    struct Header {
975        EventTime               eventTime;
976        Category                category;
977        Oid                     networkElemId;
978        AlarmLevel              networkElemAlm;
979        Oid                     maintUnitId;
980    };
981
982    // event filter (not all members valid for all categories)
983    struct Filter {
984        Category                category;
985        Oid                     networkElemId;
986        AlarmLevel              networkElemAlm;
987        Oid                     maintUnitId;
988    };
989
990    // event header (not all members valid for all categories)
991    struct Header {
992        EventTime               eventTime;
993        Category                category;
994        Oid                     networkElemId;
995        AlarmLevel              networkElemAlm;
996        Oid                     maintUnitId;
997    };
998
999    // event filter (not all members valid for all categories)
1000   struct Filter {
1001      Category                category;
1002      Oid                     networkElemId;
1003      AlarmLevel              networkElemAlm;
1004      Oid                     maintUnitId;
1005  };

```

```

        AlarmLevel      maintUnitIdAlm;
        CmdId          cmdId;
    };

5      // cookie for client registration/deregistration
      typedef long          FilterId;

10     // client registration for events
11
12     // INPUTS:
13     //      id:      SessionAppId associated with the filter
14     //      filter:   EvtDist::Filter that specifies filter
15     //                  criteria
16     //      callback: EventCB that defines deliverEvent()
17     //                  operation that will be called when an
18     //                  incoming event matches the filter
19     // RETURNS:
20     //      unique FilterId for this filter
21
22     // THROWS:
23     //      EM_INVALID_SESSION_ID
24     //      EM_INVALID_APP_ID
25     //      EM_INVALID_CATEGORY
      FilterId registerFilter(in SessionAppId id,
                               in Filter      filter,
                               in EventCB    callback)
      raises(EmapiException);

26     // client filter deregistration
27
28     // INPUTS:
29     //      id:      SessionAppId associated with the filter
30     //                  to be cancelled
31     //      filterId: FilterId of the filter to be cancelled
32
33     // THROWS:
34     //      EM_INVALID_SESSION_ID
35     //      EM_INVALID_APP_ID
36     //      EM_INVALID_FILTER_ID
      void cancelFilter(in SessionAppId      id,
                        in FilterId        filterId)
      raises(EmapiException);

40   };

41
42     // client callback for event notification
43     interface EventCB {
44
45       // deliverEvent() is called by event distribution
46       // software whenever an incoming event matches
47       // a registered filter.
48
49       // INPUTS:
50       //      header: EvtDist::Header associated with the
51       //                  incoming event
52       //      data:   SeqEventData (sequence of event data)
53       //                  associated with the incoming
54       //                  event
55
56       oneway void deliverEvent(in EvtDist::Header      header,

```

```

5                               in SeqEventData      data);

// This empty operation is called during event filter
// auditing to determine whether or not the associated
// session/application is still active. If it has
// terminated, a CORBA::SystemException will be thrown
// and subsequently caught, indicating that the associated
// filter should be removed from the filter queue.

10      oneway void check();
};

15      // typedefs for alarm data structures
20      struct AlarmRecord {
25          InstId           instId;          // unique per alarm
definition
30          MO::SeqAttrCodeValue attrValList;
};

35      typedef sequence<AlarmRecord> SeqAlarmRecord;

40      struct AlarmData {
45          ClassCode        alarmDef;
          SeqAlarmRecord   alarmRecords;
};

50      typedef sequence<AlarmData> SeqAlarmData;

55      // alarm notification type
60      enum AlarmNotifyType { ALARM_SET, ALARM_CLEAR };

65      // active alarm manager interface description
70      interface AlarmManager {
75          // active alarm filter--note that logical ids are specified
since
80          // instance information may not be available at the time
alarms are
85          // reported
90          struct AlarmFilter {
95              ClassCode        alarmDef;
              LogicalId       networkElemId;
              ClassCode        maintUnitClass;
              SeqLogicalId    maintUnitId;
              AlarmLevel      alarmLevel;
};

100         // client registration for active alarms--both initial data &
update
105         // notifications (note that only one alarm filter can be
registered
110         // with the alarm manager by each client application)
115         SeqAlarmData requestAlarms(in SessionAppId      sessionAppId,
                           in AlarmFilter      filter,
                           in AlarmCB         callback)
                           raises(EmapiException);

120         // request to change filter criteria (note that only one alarm
filter
125         // can be registered with the alarm manager by each client
application)
130         SeqAlarmData changeFilter(in SessionAppId      sessionAppId,
                           in AlarmFilter      filter,
                           in AlarmCB         callback)
                           raises(EmapiException);

```

```

5           in AlarmFilter      filter)
6           raises(EmapiException);

7   alarms // request to refresh alarm list--returns snapshot of current
8   alarm  // (note that only one alarm filter can be registered with the
9   :      // manager by each client application)
10      SeqAlarmData refreshAlarms(in SessionAppId sessionAppId)
11           raises(EmapiException);

12      // client deregistration--cancel alarm set & clear forwarding
13      to the // specified client application (note that only one alarm
14      filter can
15           // be registered with the alarm manager by each client
16           application)
17           void cancelAlarms(in SessionAppId sessionAppId)
18           raises(EmapiException);
19           // request for a list of all alarms in the entire system or
20           associated
21           // with the indicated network element--supports the EMS
22           version of the
23           // "OP:ALARM" technician command
24           CmdSeqNo opAlarm(in SessionAppId      sessionAppId,
25                           in ClassCode        alarmDef,
26                           in LogicalId       networkElemId,
27                           in EventCB         ackCB,
28                           in EventCB         cmdRespCB)
29           raises(EmapiException);
30       };

31       // client callback for active alarm notification
32       interface AlarmCB {
33           oneway void deliverAlarms(in AlarmNotifyType      type,
34                                     in AlarmData        data);
35       };

36       // active alarms service object interface, derived from managed object
37       // interface--note that no filtering of active alarms can be specified
38       interface ActiveAlarms : MO {
39           // system client (e.g., Active Alarm Manager) registration for
40           // active alarms--both initial data and update notifications
41           SeqAlarmRecord getAlarms(in SessionAppId      sessionAppId,
42                                     in AlarmCB        callback)
43           raises(EmapiException);

44           // system client (e.g., Active Alarm Manager) deregistration--
45           // cancel active alarms update notifications
46           void cancelAlarms(in SessionAppId sessionAppId)
47           raises(EmapiException);
48       };

49       #endif // _EMAPI_IDL

50       // EOF //

```

APPENDIX B-GLOSSARY

5	Alarm	The description of an alarmed notification.
10	Attribute	A property of a managed object (e.g. alarm state).
15	Attribute Code	An integer value which uniquely identifies an attribute of a given managed object.
20	Class Code	An integer value which uniquely identifies a managed object class.
25	Configuration Information	Generic term which has one of two meanings depending on its context: With respect to a managed object class, this term applies to the identification of all instances of the class, either for a specific network element or for all network elements in the system.
30	CORBA	Common Object Request Broker Architecture
35	EMAPI	Element Management Application Programming Interface
40	EMS	Element Management System
45	Event	The description of a spontaneous occurrence, such as alarm notification, command acknowledgment or configuration change.
50	Instance Identifier	An integer value which uniquely identifies an instance of a given managed object.
55	Interface Operation	Generic term for distributed service request. The target method may be defined in the Element Management Application Programming Interface (e.g. status registration) or in an application-specific derivation of a managed object (e.g. command execution).
60	Logical Identifier	An integer value which represents the logical number of a device or interface (e.g. AP 4). Note that there is no direct correlation between a logical id and instance id.
65	Managed Object	An abstract representation of a physical or logical resource which may be managed by the EMS (e.g. network element, maintenance unit, data link).
70	ORB	Object Request Broker
75	Object Identifier	The combination of managed object class code and instance identifier which

		uniquely identifies any managed object instance in the system.
5	Persistent Attribute	Information stored and kept current irrespective of any client request (e.g. maintenance state).
10	Service Object	Any EM Server object which provides services to client applications.
15	Session	Each client must establish a session at initialization--for which a unique session identifier is assigned--that will be used to validate access permissions, to correlate client requests and to keep track of Server resources utilized in behalf of any applications associated with the session.
	Status Information	Current attribute values for a managed object instance.

20 **Claims**

1. In a network having a controllable network element, a method for controlling the network element from a remote work station connectable to the network, comprising the steps of:
 - registering the network element for attributes to be tracked; and
 - polling for attributes associated with the network element only if the client requests the monitoring of the network element.
2. The method of claim 1 wherein the polling is performed by the server.
3. The method of claim 1 including the step of reporting changes in attributes when the client requests notification of changes in attributes.
4. The method of claim 1, including the steps of:
 - polling for the attributes for a plurality of clients, and
 - reporting changes in the attributes to one of the plurality of clients requesting notification of changes in the attributes.
5. The method of claim 4 wherein the step of polling includes the step of polling once for a plurality of clients that registers for the same attributes.
6. The method of claim 5 including reporting asynchronously changes in the attributes to a plurality of clients.
7. The method of claim 1 wherein the step of polling is performed in a single polling cycle when multiple clients have registered for the same attributes.
8. The method of claim 7 including the steps of
 - running an object oriented program at the remote work station to control an object associated with the controllable network element;
 - translating interface operations generated by the work station during the running of the object oriented program to corresponding translated interface operations in an object oriented language associated with the object being controlled; and
 - connecting the corresponding translated interface operations through the network to an object server to control

the object associated with the network element in accordance with the translated interface operations.

9. The method of claim 8 in which step of translating includes the steps of
 - 5 automatically determining which of a plurality of different object oriented languages is the object oriented language of the object being controlled, and
 - 10 generating translated interface operations corresponding to the interface operations from the remote workstation to the object server in accordance with the language of the object being controlled as has been automatically determined.
10. The method of claim 8 in which the object oriented program run at the remote work station is a JAVA program.
11. The method of claim 8 in which the network element is located at a node of the network and the step of translating includes the step of
 - 15 receiving the interface operations through a communication link of the network at a node separate from the node of the network element,
 - 20 translating the interface operation through the network communication link into the corresponding translated interface operations by converting the received interface operations into IPC and TCP/IP requests.
12. The method of claim 11 in which the step of connecting includes the step of transmitting the IPC and TCP/IP requests to the object server.
- 25 13. The method of claim 12 in which the step of connecting includes the step of generating the IPC and TCP/IP request through a web-based GUI.
14. The method of claim 8 in which the object server responds to the translated object requests by
 - 30 gathering information concerning the network element, and conveying the information concerning the network node that has been gathered to the remote work station by at least by one of the ways of
 - 35 dynamically generating a web-page visual display associated with the network element being controlled for interfacing with the remote work station to display the gathered information.
15. The method of claim 14 in which the step of gathering includes the step of gathering network element information concerning at least one of the items of network element information of
 - 40 a list of all active alarms,
 - a summary of system alarms, and
 - 45 a detailed indication of the status of the network element.
16. The method of claim 15 in which the step of gathering includes the step of selectively gathering all of the items of information.
- 50 17. The method of claim 8 in which the step of translating includes the step of communication with the object server through a distributed object request architecture to provide a consistent interface to the managed object that hides implementation details associated with the object manager.
18. The method of claim 17 in which the distributed object request architecture is CORBA architecture.
- 55 19. The method of claim 8 in which the CORBA architecture functions as an IPC for functions residing on the object server to eliminate the need for platform specific language for the object oriented program at the remote station.
20. The method of claim 8 in which the CORBA architecture functions as an IPC for functions residing on the object

server to provide for distribution of functionality to multiple work station processors.

21. The method of claim 8 in which communication between the network element and the object server is through use of a network management protocol.

5 22. The method of claim 1 wherein the network management protocol is the simple network management protocol.

23. The method of claim 21 including the step of obtaining system status associated with the network element by polling and auditing pursuant to the simple network management protocol.

10 24. The method of claim 21 including the step of providing real-time notification of alarm conditions at the network element through the use of network management protocol event manager.

15 25. The method of claim 21 including the step of providing command and control signals to the network element through use of simple network management protocol set operation.

26. The method of claim 8 in which

20 the object server is part of an element management server that also includes a web server, and including the step of

displaying command and alarm output information from the network element as a web browser-based display through use of the web server.

25 27. The method of claim 26 in which

the element management server also includes an executive control processor, and including the step of

30 sending the command and alarm output information from the network element that is displayed through use of the web server to the executive control processor.

28. The method of claim 12 including the steps of

35 storing network element information concerning the network at the element management server, and

selectively providing the stored network element information to a plurality of different work stations.

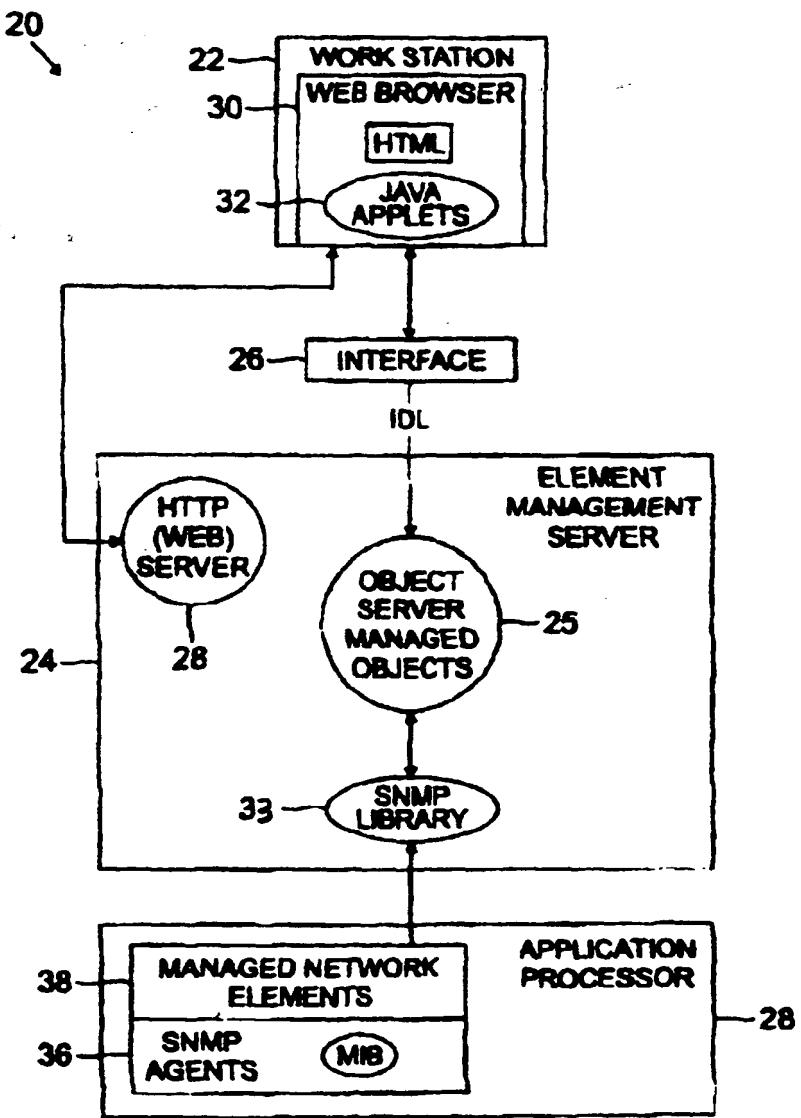
40 29. The method of claim 8 including the step of sending from the network element event and alarm notifications to the object server through use of a network management protocol.

30 30. The method of claim 29 including the step of issuing commands from the network element to obtain input information from the work station running the object oriented program to the object server through the use of the network management protocol.

45

50

55

**FIG. 1**

26 →

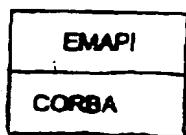


FIG. 2

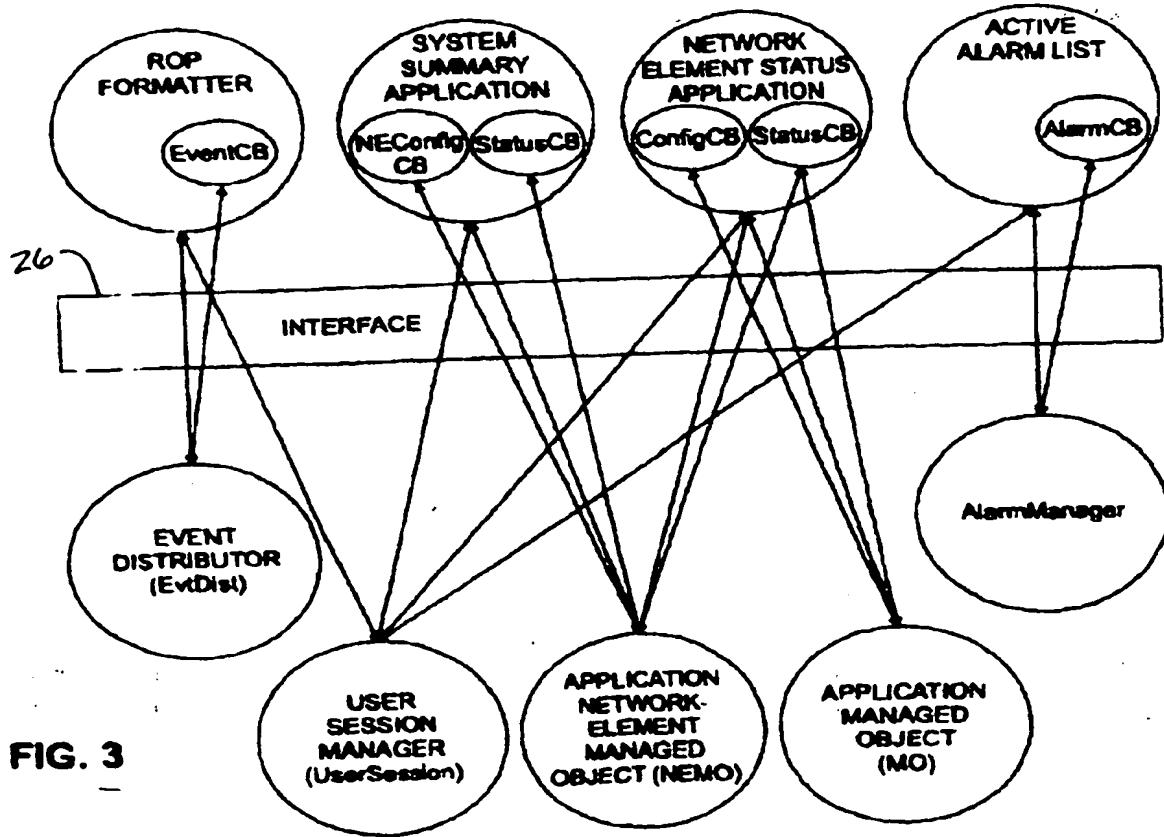


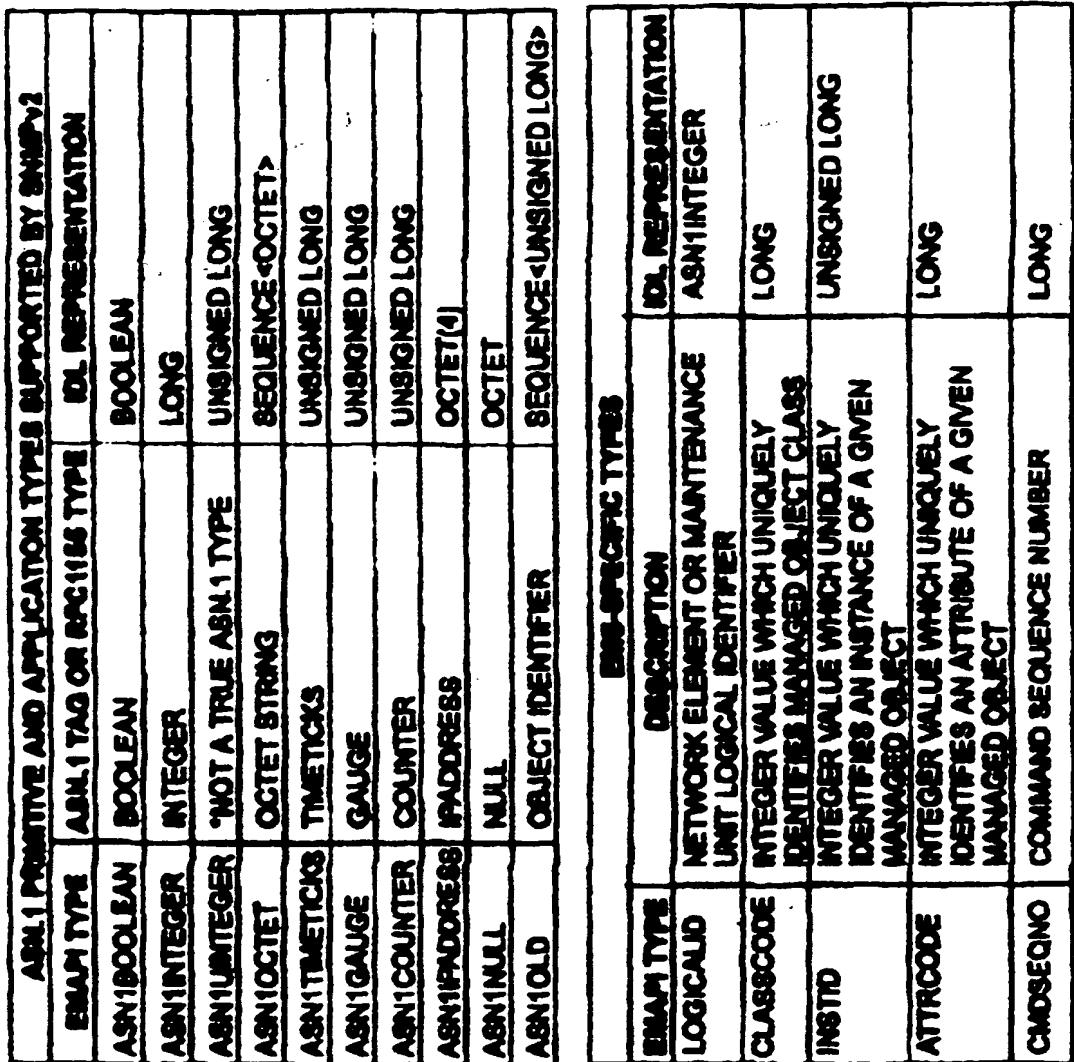
FIG. 3

User Session Manager (UserSession)	In its current form, the primary function of the user session manager is to maintain a list of active client sessions and applications. In subsequent releases, this service object will provide user access security on a network-element and operation basis. Refer to the section on Session Management for a discussion of the interfaces provided by UserSession.
Managed Object (MO)	For each physical or logical resource which must be managed by the EMS, an abstract representation will be defined which identifies attributes and operations associated with the resource. Each application-specific managed object implemented on the Server must provide the same Client interfaces for retrieving configuration information, attribute values, and registration for notification of changes. Refer to the section on Managed Objects for further details.
Network-Element Level Managed Object (NEMO)	Each application-specific NEMO implemented on the Server must provide additional interfaces above those provided by the standard managed object to support network-element level configuration queries. Refer to the section on Network Element Level Managed Objects for further details.
Event Distributor (EvtDist)	The event distributor propagates events either received or generated at the Server to clients which register filters specifying event filter criteria. Refer to the section on the Event Distributor for the definition of an event and event filter, and a discussion of the Client interfaces provided by EvtDist.
Alarm Manager (AlarmManager)	The primary utility of the AlarmManager is realized through a single client application called the AlarmList. (Note that there may be more than one instance of the AlarmList application active at any one time). Alarm filters may be registered which filter alarm information based on network element, managed object or alarm level. The AlarmManager returns an initial view of all active alarms matching the specified criteria, and provides notification of changes resulting from subsequent alarm SET or CLEAR events. Refer to the section on the Alarm Manager for the definition of an alarm and alarm filter, and a detailed discussion of the Client interfaces provided by the AlarmManager.

FIG. 4

MANAGED OBJECT CONFIGURATION CALLBACK (Config)	A Configurable object must implement a <code>deliverConfig()</code> method for notification of managed object configuration changes. Refer to the section on managed objects for details on the format of reported configuration data.
MANAGED OBJECT STATUS CALLBACK (Status)	A Status object must implement a <code>deliverStatus()</code> method for notification of managed object attribute value changes. Refer to the section on managed objects for details on the format of reported status data.
NETWORK-ELEMENT LEVEL MANAGED OBJECT CONFIGURATION CALLBACK (Messages)	An <code>INetworkElement</code> object must implement a <code>deliverConfig()</code> method for notification of network-element level managed object configuration changes. Refer to the section on network-element level managed objects for details on the format of reported network-element level configuration data.
EVENT NOTIFICATION CALLBACK (Events)	An <code>IEvent</code> object must implement a <code>deliverEvent()</code> method for event notification. Refer to the section on the event distributor for details on the format of reported event data.
ACTIVE ALARM NOTIFICATION CALLBACK (Alarms)	An <code>IAlarm</code> object must implement a <code>deliverAlarm()</code> method for notification of active alarm changes. Refer to the section on alarm manager for details on the format of reported alarm data.

FIG. 5



ASN.1 PRIMITIVE AND APPLICATION TYPES SUPPORTED BY SNMPv2

ASN.1 TYPE	ASN.1 TAG OR RFC115 TYPE	ID, REPRESENTATION
ASN1BOOLEAN	BOOLEAN	BOOLEAN
ASN1INTEGER	INTEGER	LONG
ASN1UMTEGER	'NOT A TRUE ASN.1 TYPE'	UNSIGNED LONG
ASN1OCTET	OCTET STRING	SEQUENCE<OCTET>
ASN1TIMETicks	TIME TICKS	UNSIGNED LONG
ASN1GAUGE	GAUGE	UNSIGNED LONG
ASN1COUNTER	COUNTER	UNSIGNED LONG
ASN1IPADDRESS	IPADDRESS	OCTET(4)
ASN1NULL	NULL	OCTET
ASN1OID	OBJECT IDENTIFIER	SEQUENCE<UNSIGNED LONG>

EMI SPECIFIC TYPES

ASN.1 TYPE	DESCRIPTION	ID, REPRESENTATION
LOGICALID	NETWORK ELEMENT OR MAINTENANCE UNIT LOGICAL IDENTIFIER	ASN1INTEGER
CLASSECODE	INTEGER VALUE WHICH UNQUELY IDENTIFIES A MANAGED OBJECT CLASS	LONG
INSTID	INTEGER VALUE WHICH UNQUELY IDENTIFIES AN INSTANCE OF A GIVEN MANAGED OBJECT	UNSIGNED LONG
ATTRCODE	INTEGER VALUE WHICH UNQUELY IDENTIFIES AN ATTRIBUTE OF A GIVEN MANAGED OBJECT	LONG
CMDSEQNO	COMMAND SEQUENCE NUMBER	LONG

FIG. 6

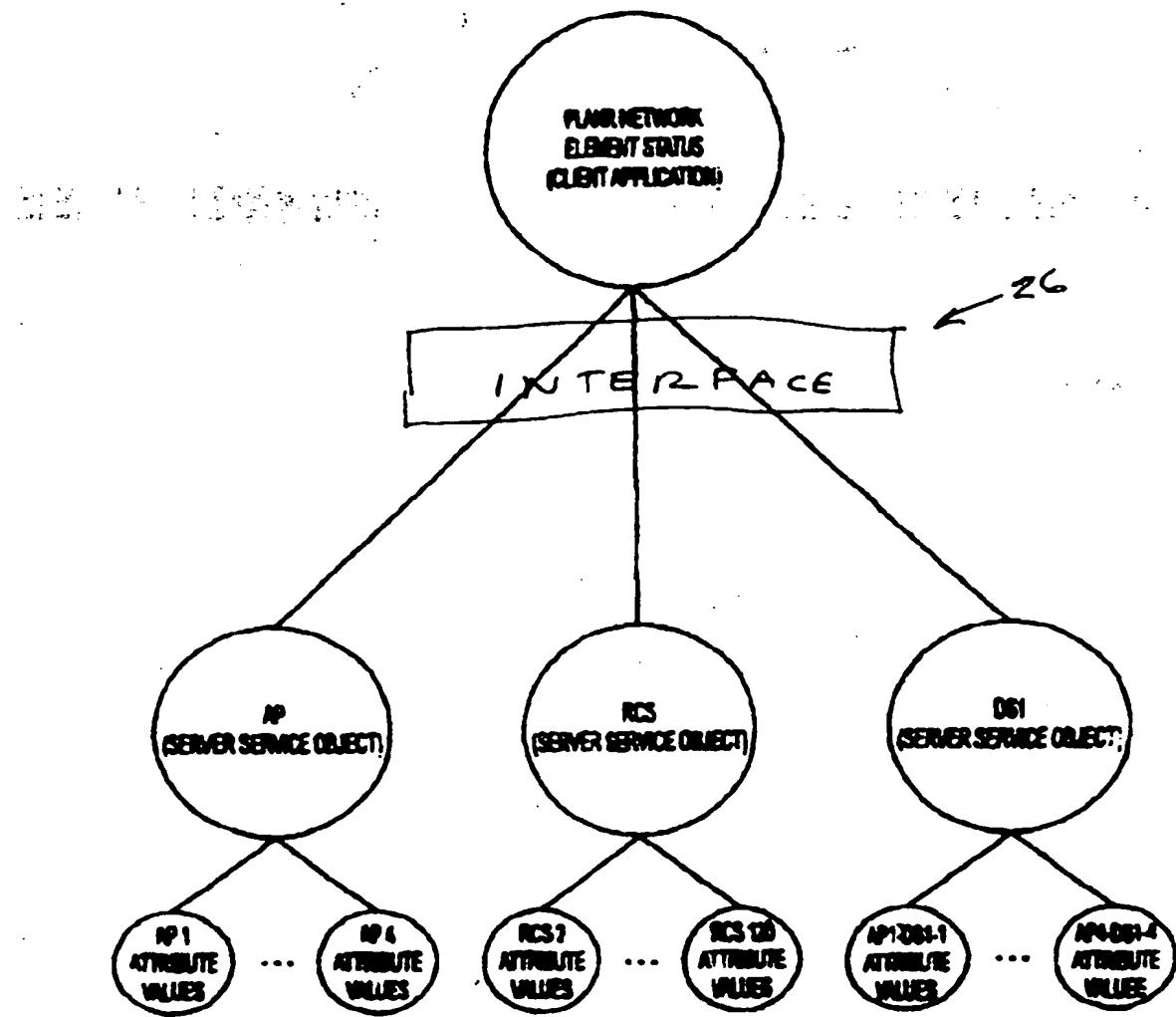


FIG.7

EVENT CATEGORY	VALID CRITERIA			
	NETWORK ELEMENT ID	NETWORK ELEMENT ALARM LEVEL	MANTENANCE UNIT ID	COMMAND ID
ALARM CLEAR	X	X	X	
ALARM SET	X	X	X	
COMMAND ACKNOWLEDGMENT	X			X
COMMAND RESPONSE	X			X
CONFIGURATION CHANGE	X		X	
INFORMATIONAL MESSAGE	X		X	
INITIALIZATION			X	
STATE CHANGE			X	X
ANY CATEGORY	X		X	X

FIG. 8

Mnemonic	Meaning
EM_INVALID_USER	The user login identifier is invalid.
EM_UNKNOWN_HOST	The specified host is unknown.
EM_TOO_MANY_SESSIONS	Too many client sessions are already active.
EM_TOO_MANY_APPLICATIONS	Too many client applications are already active for the specified login-host combination.
EM_INVALID_SESSION_ID	The specified client session id is invalid or no longer known.
EM_INVALID_APP_ID	The specified client application id is invalid or no longer known.
EM_INVALID_INST_ID	The specified instance id is invalid (e.g. Null instance is specified but not accepted in the current application context).
EM_INVALID_NE_ID	The specified network element instance id is invalid or no longer known.
EM_INVALID_MO_ID	The specified managed object instance id is invalid or no longer known.
EM_INVALID_ATTR_CODE	The specified attribute code is not defined for the target managed object.
EM_NO_MATCHING_INST	No managed object instance contains a matching key list.
EM_INVALID_FILTER	The specified filter for either an event or alarm contained one or more invalid criteria.
EM_INVALID_FILTER_ID	The specified filter id for either an event or alarm is invalid or no longer known.
EM_NE_ISOLATED	The specified network element instance is isolated.
EM_INTERNAL_ERROR	The request could not be satisfied because of an EMS Server error.
EM_INVALID_OPERATION	An invalid operation was attempted.
EM_ACCESS_DENIED	Access permission was not granted for the current operation request.
EM_VERSION_MISMATCH	Software version mismatch detected.
EM_LOST_RESOURCE	A critical resource has been lost since the last client application heartbeat (e.g. AlarmManager abnormally terminated).
EM_INVALID_KEY	An invalid key sequence was specified (e.g. wrong number of logical id's specified for a target managed object instance).
EM_INVALID_CATEGORY	An invalid event filter category was specified.

Fig. 9



European Patent
Office

EUROPEAN SEARCH REPORT

Application Number

EP 99 30 8205

DOCUMENTS CONSIDERED TO BE RELEVANT			CLASSIFICATION OF THE APPLICATION (Int.Cl.7)						
Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim							
X	US 5 491 796 A (CHEN MICHELE ET AL) 13 February 1996 (1996-02-13)	1,2,4	H04L12/26 H04L12/24						
Y	* abstract *	3,7,8, 10,17,18							
A	* figure 1 *	5,6,9, 11-16, 19-30							
	* column 2, line 43-66 *								
	* column 36, line 23 - column 37, line 64								
	*								
	* claims 1-3,7-9 *								
X	EP 0 831 617 A (DIGITAL EQUIPMENT CORP) 25 March 1998 (1998-03-25)	1,2,22							
A	* abstract *	3-21, 23-30							
	* page 2, line 51 - page 3, line 29 *								
	* figures 1-3 *								
	* claims 1,4,5,7,8 *								
Y	MAGEDANZ T ET AL: "INTELLIGENT AGENTS AN EMERGING TECHNOLOGY FOR NEXT GENERATION TELECOMMUNICATIONS?" PROCEEDINGS OF INFOCOM, US, LOS ALAMITOS, IEEE COMP. SOC. PRESS, vol. CONF. 15, 1996, pages 464-472, XP000621308 ISBN: 0-8186-7293-5	3,7,8, 10,17,18	TECHNICAL FIELDS SEARCHED (Int.Cl.7) H04L						
A	* abstract *	1,2,4-6, 9,11-16, 19-30							
	* paragraphs '0002!, '03.4!, '06.1! *								
	* figures 2,4,5 *								

<p>The present search report has been drawn up for all claims</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 33%;">Place of search</td> <td style="width: 33%;">Date of completion of the search</td> <td style="width: 33%;">Examiner</td> </tr> <tr> <td>THE HAGUE</td> <td>7 February 2000</td> <td>Cichra, M</td> </tr> </table>				Place of search	Date of completion of the search	Examiner	THE HAGUE	7 February 2000	Cichra, M
Place of search	Date of completion of the search	Examiner							
THE HAGUE	7 February 2000	Cichra, M							
<p>CATEGORY OF CITED DOCUMENTS</p> <p>X : particularly relevant if taken alone Y : particularly relevant if combined with another document of the same category A : technological background O : non-written disclosure P : intermediate document</p> <p>T : theory or principle underlying the invention E : earlier patent document, but published on, or after the filing date D : document cited in the application L : document cited for other reasons B : member of the same patent family, corresponding document</p>									



European Patent
Office

EUROPEAN SEARCH REPORT

Application Number
EP 99 30 8205

DOCUMENTS CONSIDERED TO BE RELEVANT			CLASSIFICATION OF THE APPLICATION (Int.Cl.7)
Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	
A	<p>KOTSCHEINREUTHER J: "BETREIBER BRAUCHEN OFFENE NETZMANAGEMENTSYSTEME" NTZ NACHRICHTENTECHNISCHE ZEITSCHRIFT, DE, VDE VERLAG GMBH. BERLIN, vol. 50, no. 5, 1 January 1997 (1997-01-01), pages 50-52, XP000693391 ISSN: 0027-707X * the whole document *</p> <p>-----</p> <p>"DESIGN FOR A SIMPLE NETWORK MANAGEMENT PROTOCOL SUBAGENT FOR INTERNET FIREWALLS" IBM TECHNICAL DISCLOSURE BULLETIN, US, IBM CORP. NEW YORK, vol. 40, no. 3, 1 March 1997 (1997-03-01), pages 63-68, XP000694517 ISSN: 0018-8689 -----</p>	1-10, 17, 18	1-4, 7, 22
			TECHNICAL FIELDS SEARCHED (Int.Cl.7)
	The present search report has been drawn up for all claims		
Place of search	Date of completion of the search	Examiner	
THE HAGUE	7 February 2000	Cichra, M	
CATEGORY OF CITED DOCUMENTS			
<p>X : particularly relevant if taken alone Y : particularly relevant if combined with another document of the same category A : technological background O : non-written disclosure P : intermediate document</p> <p>T : theory or principle underlying the invention E : earlier patent document, but published on, or after the filing date D : document cited in the application L : document cited for other reasons & : member of the same patent family, corresponding document</p>			

ANNEX TO THE EUROPEAN SEARCH REPORT
ON EUROPEAN PATENT APPLICATION NO.

EP 99 30 8205

This annex lists the patent family members relating to the patent documents cited in the above-mentioned European search report.
The members are as contained in the European Patent Office EDP file on
The European Patent Office is in no way liable for these particulars which are merely given for the purpose of information.

07-02-2000

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 5491796 A	13-02-1996	AU 5404194 A WO 9410625 A	24-05-1994 11-05-1994
EP 0831617 A	25-03-1998	NONE	

For more details about this annex : see Official Journal of the European Patent Office, No. 12/82

EPO FORM P049

**This Page is Inserted by IFW Indexing and Scanning
Operations and is not part of the Official Record**

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- BLACK BORDERS**
- IMAGE CUT OFF AT TOP, BOTTOM OR SIDES**
- FADED TEXT OR DRAWING**
- BLURRED OR ILLEGIBLE TEXT OR DRAWING**
- SKEWED/SLANTED IMAGES**
- COLOR OR BLACK AND WHITE PHOTOGRAPHS**
- GRAY SCALE DOCUMENTS**
- LINES OR MARKS ON ORIGINAL DOCUMENT**
- REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY**
- OTHER:** _____

IMAGES ARE BEST AVAILABLE COPY.

As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.

THIS PAGE BLANK (USPTO)